Efficient Parallelization of a Three-Dimensional Navier-Stokes Solver on MIMD Multiprocessors

To appear in Journal of Supercompuitng

A. Averbuch B. Epstein
School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978

L. Ioffe I. Yavneh
Faculty of Computer Science, Technion, Haifa 32000, Israel

Abstract

We present an efficient parallelization strategy for speeding up the computation of an high-accuracy 3-dimensional serial Navier-Stokes solver that treats turbulent transonic high-Reynolds flows. The code solves the full compressible Navier-Stokes equations and is applicable to realistic large size aerodynamic configurations and as such requires huge computational resources in terms of computer memory and execution time. The solver can resolve the flow properly on relatively coarse grids. Since the serial code contains a complex infrastructure typical for industrial code (which ensures its flexibility and applicability to complex configurations), then the parallelization task is not straightforward. We get *scalable* implementation on massively parallel machines by maintaining efficiency at a fixed value by simultaneously increasing the number of processors and the size of the problem.

The 3-D Navier-Stokes solver was implemented on three MIMD message-passing multiprocessors (a 64-processors IBM SP2, a 20-processors MOSIX, and 64-processors Origin 2000). The same code written with PVM and MPI software packages was executed on all the above distinct computational platforms. The examples in the paper demonstrate that we can achieve efficiency of about 60% for as many as 64 processors on Origin 2000 on a full-size 3-D aerodynamic problem which is solved on realistic computational grids.

1 Introduction

Flow problems of Computational Fluid Dynamics (CFD) and associated scientific and engineering disciplines such as Computational AeroElasticity (CAE), Computational AeroAcoustics (CAA) are notoriously difficult to solve. In the last decade this technique has been implemented in industry for an increasing number of complex problems such as advanced aerodynamic design of aircrafts or suppression of jet exhaust noise for high-speed planes in aircraft industry, blood flow simulation in medical environment, to mention only three examples.

Because of the above reasons, practical usage of existing serial Navier-Stokes solvers is limited to either coarse computational grids or geometrically simplified configurations. This is especially true in the case of high Reynolds (Re) turbulent flows in transonic regime. Even so, the corresponding computer runs are mostly too computational intensive to allow the acquisition of results on a daily basis needed in industry. Moreover, the direct numerical simulation (DNS), i.e. solution of the complete Navier-Stokes equations which are not based on a turbulence modeling or even the implementation of more elaborated models of turbulence is clearly beyond the ability of even the fastest serial supercomputers. (The computational work for DNS problems is roughly proportional to Re^3 and modern serial or vector computers are able to perform simulations with Reynolds numbers of several hundreds only while aircraft industry applications, for example, deal with Re numbers that range from 10^5 to 10^8). The above figures also indicate that the anticipated progress in high-speed serial computing alone will not solve the problem in the long run. Additional dimension to the computational framework is provided by the availability of scalable parallel systems with large number of processors. These systems will be beneficial to these types of problems only if there are efficient and scalable parallel algorithms.

The difficulties of the proposed problem are generated because the mathematical basis, viz., the Navier-Stokes Equations, is a sophisticated non-linear time-dependent system of partial differential equations (PDE) with a complicated spectrum. Therefore, high resolution is needed to cover the inherently wide range of scales. High accuracy is required in most realistic important problems (for example, to get a realistic estimate of the drag coefficient, even for an isolated aerodynamic wing, at least 6 or 7 significant digits are required). In addition, existing numerical algorithms suffer from slow rates of convergence and as realistic problems possess complex and irregular geometry distinct complex numerical meshes are needed, and additional difficulties and problems arise in the management of a set of grids, including grid-overlaps. These extra difficulties require additional computational resources.

This work is motivated by the need for the next generation solvers to employ algorithms and software that are able to handle, efficiently and accurately, realistic physical models. They must provide numerical solutions to the Navier-Stokes equations with turbulence models. They must be able to (a) achieve high accuracy; (b) treat realistic geometries; (c) operate with computational efficiency (including parallel implementation) that will allow solving realistic problems with available resources. To meet these demands the solver must combine a modular treatment of the geometry, in order to treat complex realistic configurations, with a highly accurate numerical approach with good convergence properties.

In this work, an efficient parallelization strategy for speeding up the computation time of an existing high-accuracy 3-dimensional serial Navier-Stokes solver that treats turbulent transonic high-Reynolds flows is proposed. A serial industrial code called NES, originally developed at the Israel Aircraft Industries (IAI), was chosen to be parallelized. The code solves the full compressible Navier-Stokes equations and is applicable to realistic large size aerodynamic configurations and as such requires huge computational resources in terms of computer memory and execution time. This happens in spite the fact that its overall cost-effectiveness is favorable when it is compared to other available Navier-Stokes codes. This is due to its ability to resolve the flow properly on relatively coarse grids. To allow the practical use of the code on a daily basis it is vital to parallelize it in order to reduce substantially the computational resources the solver necessitated. In practice, an efficient parallel scheme has to be developed in order to tailor it to the structure of the serial Navier-Stokes code, in order to reduce the communication overhead and thus to achieve reasonably high scalability of the code. Since the NES serial code has a complex infrastructure typical for industrial code (which ensures its flexibility and applicability to complex configurations), then the parallelization task is not straightforward. The goal is to get *scalable* implementation on massively parallel machines by maintaining efficiency at a fixed value by simultaneously increasing the number of processors and the size of the problem. The *scalability* of a parallel system is a measure of its capacity to increase speedup in proportion to the number of processors. It reflects a parallel system's ability to utilize increasing processing resources effectively.

The 3-D Navier-Stokes solver was implemented on three MIMD message-passing multiprocessors (a 64-processors IBM SP2, a 20-processors MOSIX[1], and 64-processors Origin 2000). The same code written with PVM (parallel virtual machine[11]) and MPI software packages was executed on all the above distinct computational platforms. The examples in the paper demonstrate that we can achieve efficiency about 60% for as many as 64 processors on Origin 2000 on a full-size 3-D aerodynamic problem which is solved on realistic computational grids.

The mathematical background with a description of the serial NES code is given in section 2. The parallel implementation is given in section 3. The achieved parallel performance is discussed in section 4.

2 Mathematical Background and Code Description

2.1 Numerical Approach

To achieve high aerodynamic accuracy (the ability to estimate sensitive aerodynamic characteristics such as lift, pressure and friction drag, pitching moment) the present method [6],[7],[10] uses a numerical approach which does not employ any explicit artificial dissipation. The algorithm is based on the Essentially Non-oscillatory (ENO)[12] concept and it is applied via flux interpolation technique presented in [15]. It is implemented in physical space by finite volume method on grids which were defined by a set of vertices and it is applicable to reasonably smooth computational grids which are not necessarily defined by mapping functions.

Nonlinear stability is maintained by approximating inviscid fluxes on a variable template according to local characteristics and smoothness of the fluxes (which may change with iterations); viscous fluxes are approximated in a straightforward way. A 3-stage Runge-Kutta scheme is applied in a Total Variation Diminishing (TVD) preserving form. Specially constructed algorithm is used to considerably diminish the amount of computational work associated with changeability of template, and to accelerate the convergence to steady-state.

The Navier-Stokes equations in Cartesian coordinates may be written in the form:

$$\mathbf{q}_t + div\mathbf{C} = div\mathbf{V} \tag{1}$$

where the tensor $\mathbf{C} = (\mathbf{f}, \mathbf{g}, \mathbf{h})$ represents the convection terms, the tensor $\mathbf{V} = (\mathbf{r}, \mathbf{s}, \mathbf{h})$ represents the viscous terms, $\mathbf{q} = (\rho, \rho\mathbf{u}, \rho\mathbf{v}, \rho\mathbf{w}, E)$ is the density, $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ is the velocity vector, E is the energy, t is the time, $\mathbf{f}, \mathbf{g}, \mathbf{h}$ are the inviscid (convection) fluxes and $\mathbf{r}, \mathbf{s}, \mathbf{t}$ are the viscous fluxes which depend in a nonlinear mode on $\mathbf{q}$.

We assume that the grid consists of cells that form an (i, j, k) structure. By integrating over each cell separately we get a system of ODE's which can be solved by a time-stepping procedure.

For a particular cell (i, j, k) the following approximation is assumed:

$$(\Omega_{i,j,k}\mathbf{q}_{i,j,k})_t + [\mathbf{C}\cdot(S\mathbf{n})]_{i+0.5,j,k} - [\mathbf{C}\cdot(S\mathbf{n})]_{i-0.5,j,k} + [\mathbf{C}\cdot(S\mathbf{n})]_{i,j+0.5,k} - [\mathbf{C}\cdot(S\mathbf{n})]_{i,j-0.5,k} +$$

$$[\mathbf{C}\cdot(S\mathbf{n})]_{i,j,k+0.5} - [\mathbf{C}\cdot(S\mathbf{n})]_{i,j,k-0.5} = [\mathbf{V}\cdot(S\mathbf{n})]_{i+0.5,j,k} - [\mathbf{V}\cdot(S\mathbf{n})]_{i-0.5,j,k} + [\mathbf{V}\cdot(S\mathbf{n})]_{i,j+0.5,k} -$$

$$[\mathbf{V}\cdot(S\mathbf{n})]_{i,j-0.5,k} + [\mathbf{V}\cdot(S\mathbf{n})]_{i,j,k+0.5} - [\mathbf{V}\cdot(S\mathbf{n})]_{i,j,k-0.5} \tag{2}$$

where $\Omega_{i,j,k}$ is the cell volume, $\mathbf{q}_{i,j,k}$ is some mean value of $\mathbf{q}$ over the cell and S is the area of a cell side surface. Fluxes with half-indices are approximated by a one-dimensional interpolation from nearby cell centers. That is, fluxes with subscripts $i + \frac{1}{2}, j, k$ and $i - \frac{1}{2}, j, k$ are approximated by fluxes at the cell centers with the same j and k and so on. A difficulty is introduced by the use of a staggered grid (with $\mathbf{q}, \mathbf{C}$ defined at cell centers and the associated fluxes on the cell sides, as opposed to the case of grids defined by an analytic mapping where the curvilinear fluxes are well defined at the cell centers). The interpolation operator I must possess the following properties:

Approximation: in the case of a smooth grid the interpolation operator I approximates the corresponding curvilinear flux in a mapping-defined system of coordinates.

Consistency: the interpolation operator I has to be exact for a constant-field solution.

The above properties are ensured, e.g., by choosing the operator I in the form

$$[\mathbf{C}\cdot(S\mathbf{n})]_{i+0.5,j,k} = I\{[\mathbf{C}_{l,j,k}\cdot(S\mathbf{n})]_{i+0.5,j,k}\} \quad l = i, i+1, i-1, \ldots \tag{3}$$

where the direction vector frozen in the interpolant.

Fluxes in the RHS of Eq. (3) are projected into local characteristic fields. The ENO template of the RHS of Eq. (3) is determined separately in each field primarily according to the sign of the corresponding eigenvalue and then according to the smoothness of the projected fluxes. This template can change dynamically through the iterations. Finally, interpolated characteristic fluxes are projected back to get the Cartesian ones.

Finite differences are used to approximate the first derivatives along the local grid coordinates. The Cartesian derivatives are obtained by combining the above derivatives in a way which ensures exactness for a linearly changing flow-field. A 3-stage Runge-Kutta scheme is applied in a TVD preserving form [15]. To accelerate the convergence to steady-state, explicit residual smoothing is applied allowing CFL numbers of about 1.5 for subsonic and transonic flow. For steady-state computations, local time steps are used based on local eigenvalues calculated separately for inviscid and viscous operators. Usually no other parameters are involved in the time-marching procedure.

The variable template determination is highly time-consuming, therefore, the following template determination strategy was implemented for subsonic and transonic cases:

1. New templates are determined only once for all three Runge-Kutta stages.

2. A linearly stable template is applied everywhere but in the immediate neighborhood of "sensitive" cell faces where the full variable ENO template determination procedure is applied (the cell face is called sensitive if one of the sonic eigenvalues has opposite signs across the face).

The neighborhood is determined by the number of points in the ENO template minus 1 so that sensitivity check is easy to perform. In addition, there is considerable computer time savings and the procedure enhances the numerical accuracy.

The original serial code is embedded in the framework of a multigrid algorithm [4, 5]. This means that the grids are united in levels arranged in a hierarchical way which starts from a coarse level and then refine it so that the coarser level grids contain the next level refinement geometrically. Local refinements are allowed and used in practical computations.

2.2 Convergence and Accuracy

ENO schemes maintain stability by varying the computational template where necessary. As a result, the maximum residuals norm is no longer appropriate for the convergence estimate purposes. Instead, the convergence is estimated, together with the reduction of residuals in some integral norm by several orders of magnitude, by the steadiness of the solution (negligibility of variations in the computed flow characteristics) which also reflects the ability of the grid to resolve flow properly. New improvements to the method allowing better (multigrid) convergence are discussed in [10].

2.3 Sample results by the serial code NES

Solutions for viscous transonic flow over two different wing configurations are presented. C-O grid topologies are used with normal spacing at the wing surface ensuring $y+$ of about 1.5-2 on fine meshes where $y+$ is the nondimensional measure of the boundary layer resolution. The effects of turbulence are modeled through the Baldwin-Lomax model [3].

2.3.1 ONERA-M6 wing

The computed solution for this popular transonic test-case is in close agreement with experiment at $M_\infty = .84$, $\alpha = 3.06^\circ$ along the whole wing span. Pressure coefficients are given in Fig. 1. The computed lift coefficient and drag coefficient values are almost identical to the results of [14, 16] where the grids contained about 1 million grid points compared to slightly less than 100000 points used by the NES code.

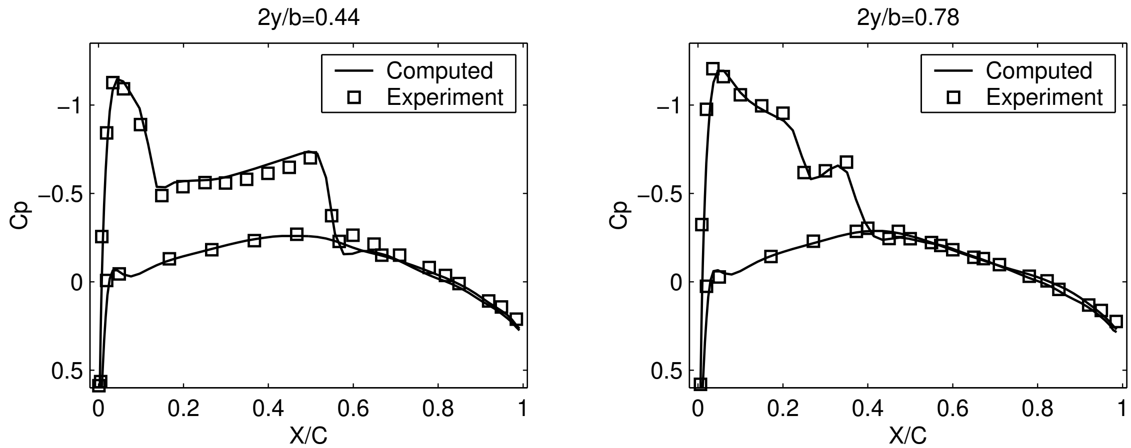

Figure 1: ONERA-M6. Chordwise C_p distribution at $M_\infty = 0.84,\ \alpha = 3.06°$ at $2y/b = 0.44$ and 0.78

2.3.2 Transport-type cranked wing

Here the transonic flow-field is highly influenced by shock-boundary layer interaction. The details concerning the geometry of this twisted wing with highly cusped profiles can be found in [8, 9]. The grids contain up to 100000 points. The Reynolds number is 3.0×10^6. Typical aerodynamic coefficients are compared with experiment in Fig. 2 and pressure comparisons are shown in Fig. 3. For details and the discussion of results see [6, 8].

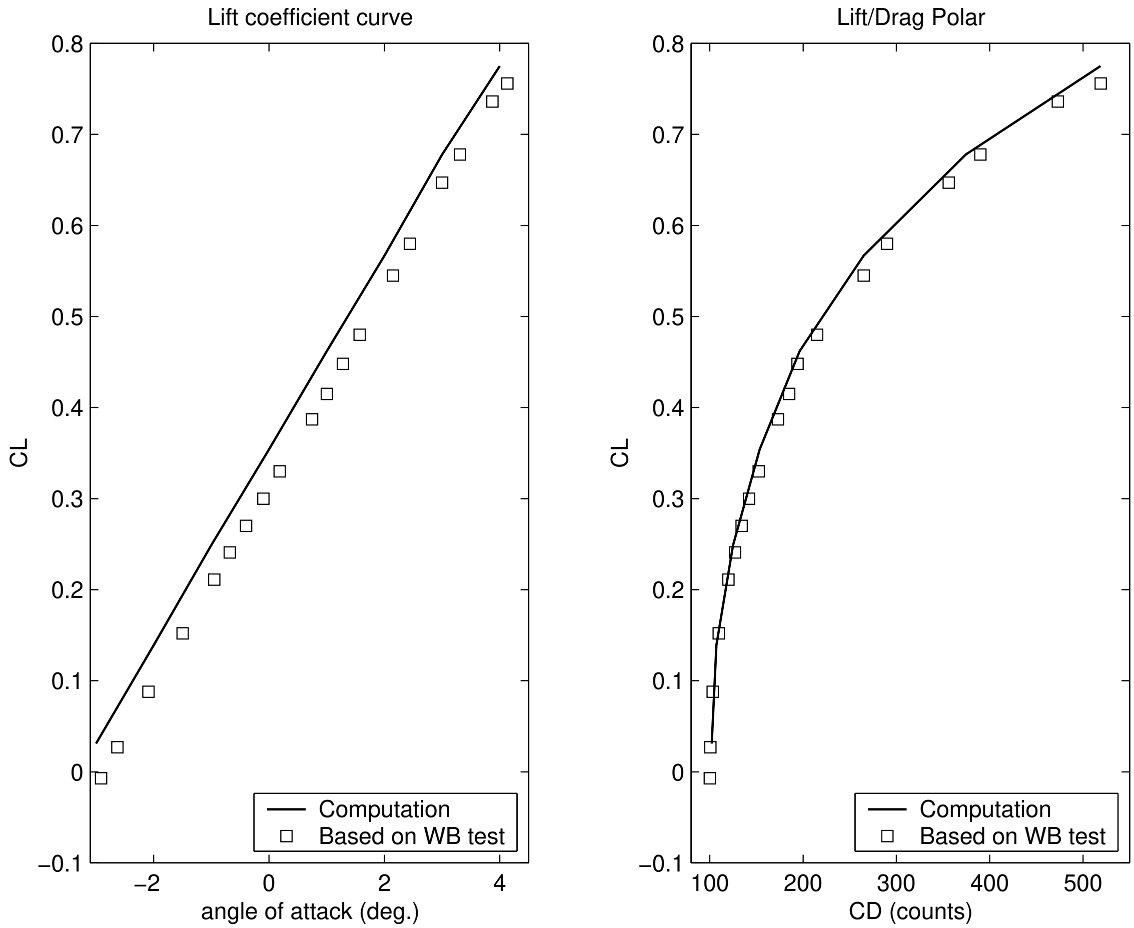

Figure 2: Transport-type wing. Lift coefficient curve and lift/drag polar at $M_\infty = 0.75$, $Re = 3.5 \times 10^6$

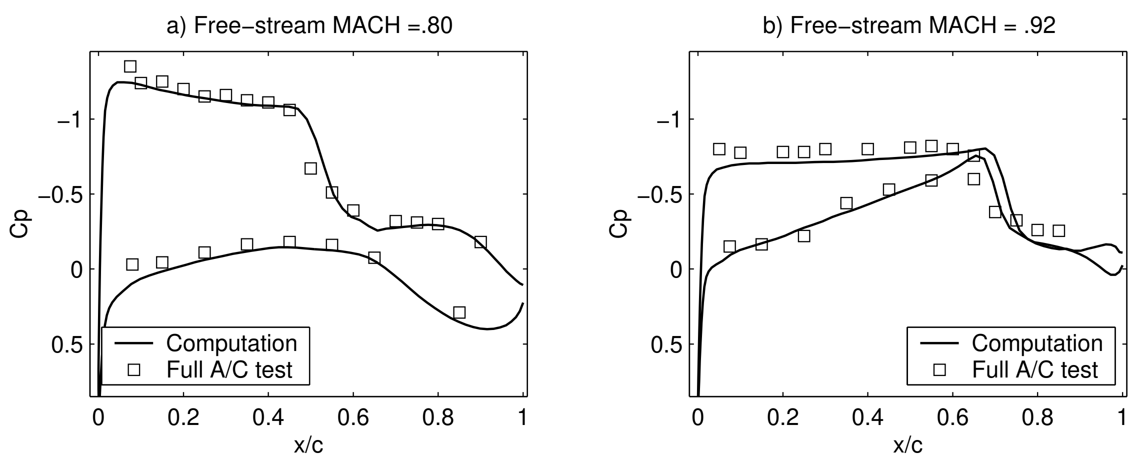

Figure 3: Transport-type wing. Chordwise C_p distribution at $2y/b = 0.65$ at fixed angle of attack, $Re = 3.5 \times 10^6$; a) $M_\infty = 0.80$. b) $M_\infty = 0.92$.

3 The Parallel Algorithm

3.1 Choice of Parallelization Strategy

No central manager (like in Server-Client) was chosen as a model for parallelization due to the facts that CFD problems are large and the serial algorithm uses (mostly) explicit relaxation with a relatively small template. Therefore, a complete distributed model is implemented.

3.2 Preprocessing: Data Partition

The proposed parallelization scheme is based on domain decomposition. Basically, it consists of constructing a set of computational subgrids which were derived from the initial serial grid in order to distribute the computational work among the concurrent processors. These subgrids are referred later as blocks. In the present case the initial (serial) grids are united in levels in such a way that coarser level grids contain the next refinement level. Therefore, it is necessary to incorporate a decomposition strategy into the multigrid structure of the data, i.e. each grid that participates in the computation must be subdivided in a way which ensures the correct structure of the appropriate multigrid tree. For example, consider grid 202 on the multigrid level 2 which geometrically contains a finer grid 304 from the next level 3. Then, we can assume that decomposition of grid 202 of level 2 contains blocks 202A, 202B while grid 304 of level 3 contains the blocks 304A, 304B, 305C. These decompositions must preserve the original multigrid structure: each of the blocks 304A, 304B, 304C have to be fully contained in either block 202A or 202B depending on the specific geometric partition. In addition, adjacent blocks which were derived from the same grid (i.e. 202A and 202B), must overlap in order to ensure the availability of the proper numerical template for all the computational points. This dictates the requirement for the information exchange among corresponding processors.

In fact, in our case we split the initial *multigrid tree* into trees of blocks according to the number of available processors and the given decomposition. The numerical scheme is then applied on each partial tree provided that the necessary information is supplied to ensure the correct overlapping of adjacent blocks. The important feature of the above decomposition is that only neighbouring blocks "talk (communicate) to each other" without the need to transfer a global information[1].

The above preprocessing procedure is performed only once for the specific multigrid configuration.

3.3 Code Modification - Structural Changes

The serial code was modified in such a way that each processor was able to process its own set of blocks which are united to form a multigrid subtree. The numerical scheme is then independently applied on a current subset of blocks (corresponding to the multigrid level being processed) in each processor. At the end of each computation cycle which represents one Runge-Kutta stage - see section 2, part of the newly calculated data is transferred to

[1]Global communication can degrade substantially the achieved speedup

neighbouring processors thus allowing a set of multigrid subtrees to perform exactly as in the serial set of grids (originally organized as a multigrid tree). The size of the information to be transferred remains constant (that is , it does not change with iterations) and depends on the specific choice of the decomposition directions; this will be discussed in subsection 3.5. In the simplest case, when, for example, decomposition in the $z-$direction is employed, the amount of information which is transferred is proportional to $N_x \cdot N_y$, where N_x and N_y are block dimensions in the x and y directions, respectively.

3.4 The Expected Scalability from the Use of the Parallel Algorithm

Initially, we consider a general explicit finite-difference (or finite-volume) algorithm which is applied to a PDE problem on a structured grid that has to be parallelized on massively parallel systems with distributed memory. Suppose we are given a single grid with $i-j-k$ structure that contains $N_i, \quad i = 1,2,3$ cells in each direction, respectively. We get total of $N_t = N_1 \times N_2 \times N_3$ computational cells. The expected amount of computational work (at least for pure explicit numerical schemes) is proportional to the number of computational cells. Therefore, it is natural to decompose the computational grid into cells of equal volumes. If the decomposition is performed on some of the i, j, k directions (the actual set of indices is denoted by P) then each subgrid contains N_t/C_t cells, $C_t = \prod_{i \in P} C_i$ where C_i is the partition factor in the i direction. Thus, the work will be distributed among up to C_t parallel processors with distributed memory where each processor is assigned subdomain. The expected amount of computational work on each processor is proportional to $W_{comp} = N_t/C_t$.

In order to get identical results between the serial and the parallel computation it is necessary to exchange information among neighbouring grids after each iteration step.

The information to be transferred among processors depends on the width of the computational template and the topology. It is possible to estimate the amount of information which is transferred from a given processor in the $i, \quad i \in P$ direction by

$$R_i = \frac{N_t}{C_t} \frac{C_i}{N_i} d_i$$

where d_i depends on the above mentioned factors (template, grid topology).

Most practical decompositions for explicit finite-difference schemes ensure the following properties:

1. The width of the computational template (and consequently the width of transferable strips) is a small number which ensures that the volume of the participating information in the communication is essentially less than the computational volume.

2. Each processor usually communicates with not more than two neighbours in each direction.

The two properties assure that the factor d_i is proportional to the size of the template for each i.

The required time for a single session of concurrent communications equals to

$$W_{comm} = \sum_{i \in P} \frac{R_i}{\sigma(R_i)}$$

where σ is the throughput function (information per a time unit) for a specific message passing. $\sigma \leq \sigma_{max}$. Thus, W_{comm} is estimated by

$$\begin{array}{lr} W_{comm} \geq \sum_{i \in P} \left(\frac{R_i}{\sigma(R_i)}\right) \geq \sum_{i \in P} \left(\frac{R_i}{\sigma_{max}}\right) & = \\ \left(\frac{1}{\sigma_{max}}\right) \sum_{i \in P} \left(\frac{N_t}{C_t}\frac{C_i}{N_i} d_i\right) & = \\ \frac{N_t}{C_t \sigma_{max}} \sum_{i \in P} \left(\frac{C_i}{N_i} d_i\right) & \geq \\ const_1 C_t^{-1} \left(\prod C_i\right)^{\frac{1}{D}} \left(\prod d_i\right)^{\frac{1}{D}} & = \\ const_2 C_t^{\frac{1}{D}-1} & \end{array}$$

where D is the number of directions in the decomposition and $const_1$, $const_2$ do not depend on the number of parallel processors C_t. Finally, since $W_{comp} = N_t / C_t$ then

$$W_{comm} \geq const \left(W_{comp}\right)^{1-\frac{1}{D}}$$

where the constant is independent of the number of processors. It clearly depends on the dimensions of the original serial problem, the throughput function and the set P of indices.

From the last inequality it is simple to derive

1. In case $D = 1$, which corresponds to one-dimensional decomposition, the problem is not scalable since W_{comm} is bounded from below by W_{comp} independently from the number of processors.

2. For a 2-dimensional decomposition ($D = 2$) or for $D = 3$ (where the decomposition is performed in all the three directions) the problem exhibits only a limited scalability.

In the best case analysis, W_{comm} behaves as $W_{comp}^{\frac{1}{2}}$ for $D = 2$ and $W_{comp}^{\frac{2}{3}}$ for $D = 3$ provided that:

a. Domain decomposition is close to the optimal. It follows from a previous consideration that the optimal decomposition follows the rule

$$\frac{C_{dir1}}{C_{dir2}} = \frac{N_{dir1}}{N_{dir2}}$$

for every two directions of decomposition.

b. High level of throughput is maintained.

c. Additional communications overhead associated with loops organization and similar factors (and partly reduced by an improved use of cache for a diminished size problem) is relatively low.

Consider the efficiency factor

$$E = \frac{W_{comp}}{W_{comp} + W_{comm}}$$

for the case $D = 2$. When the efficiency is about 80% on 16 processors, we conclude that the theoretical efficiency of 74% maybe achieved for 32 processors and above 66% for 64 processors if the conditions (a),(b), and (c) are maintained.

3.5 Directions of decomposition

The assumptions of subsection 3.4 are implemented here for a specific grid topology with the numerical method which was described in section 2. Grids which were employed in the (serial) code possess C-O topology and i, j, k structure. The structure of a typical prototype grid is shown in Fig. 4.

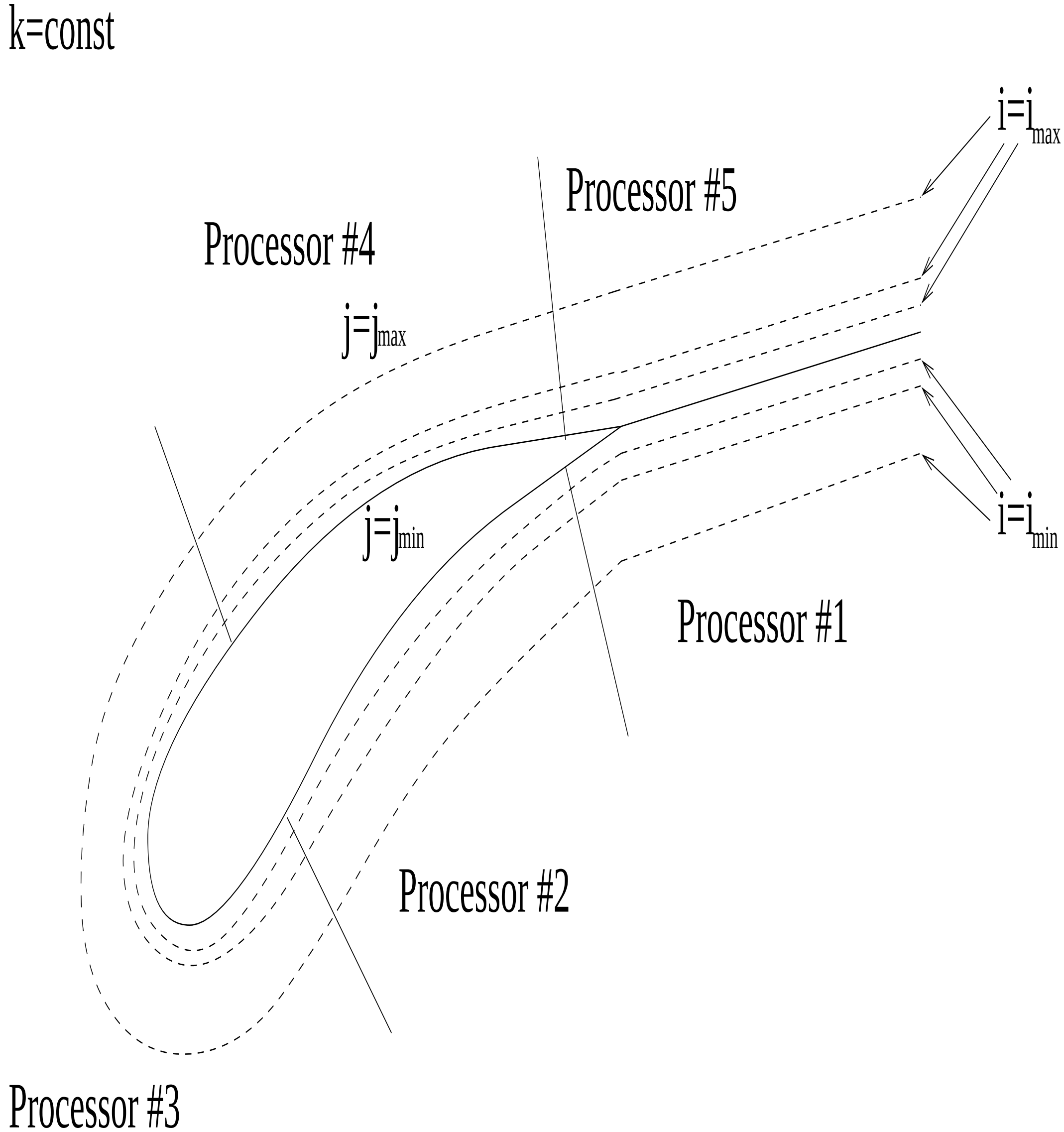

Figure 4: Domain decomposition in the streamwise i direction at $k = const$.

Specific serial grids are derived from the finest grid which was constructed for a given geometry. They are derived by the process of coarsening and (optional) partitioning in one or more directions i, j and k preserving the i, j, k structure. Thus, a natural decomposition intended for parallelization purposes must follow this pattern. Following the conclusions of subsection 3.4 it is preferable to decompose each grid into its three directions in order not to degrade the achieved speedup and scalability. In the present application the decomposition in the $j-$ direction is not used for the following reason: the Baldwin-Lomax turbulence model employed in the code for the solution of the Navier-Stokes equations uses the integral information along the whole $j-$ coordinate line so the use of the $j-$decomposition implies a heavy communications consumption (the assumptions in subsection 3.4 are not valid anymore for the $j-$direction). Instead of having 3 decomposition directions (i, j and k) we used only the i and k directions for all the grids. The choice of the partition factors is dictated by the following considerations:

1. The proportionality rule given in subsection 3.4.

2. A restriction implied by the grid topology. This restriction applies to the case where the number of computational cells in the $i-$direction assigned to each processor (N_1/C_1) is less than the number of cells in the wing wake (the region beyond the trailing edge of the wing, see Fig. 4). In this case the processors which were assigned to the wake area have an additional neighbour to communicate with. This increases the coefficient d_1 and the total amount of communications.

3. The use of blocks which are too "thin" (relatively to the width of the computational stencil) increases the number of processors participating in the communication process and may use processors which are not classified as immediate neighbors.

The optimal choice of the partition parameters is a result of the trade-off between the demands of (1), (2) and (3) above.

The following schematic examples illustrate the implementation of the above principles for decomposition. First, a possible decomposition in the $i-$direction is shown (at $k = const$) in Fig. 4, where each processor which participates in the computation, exchanges information with at most two adjacent processors. An example for decomposition in the $k-$direction for a full multigrid problem is illustrated in Fig. 5.

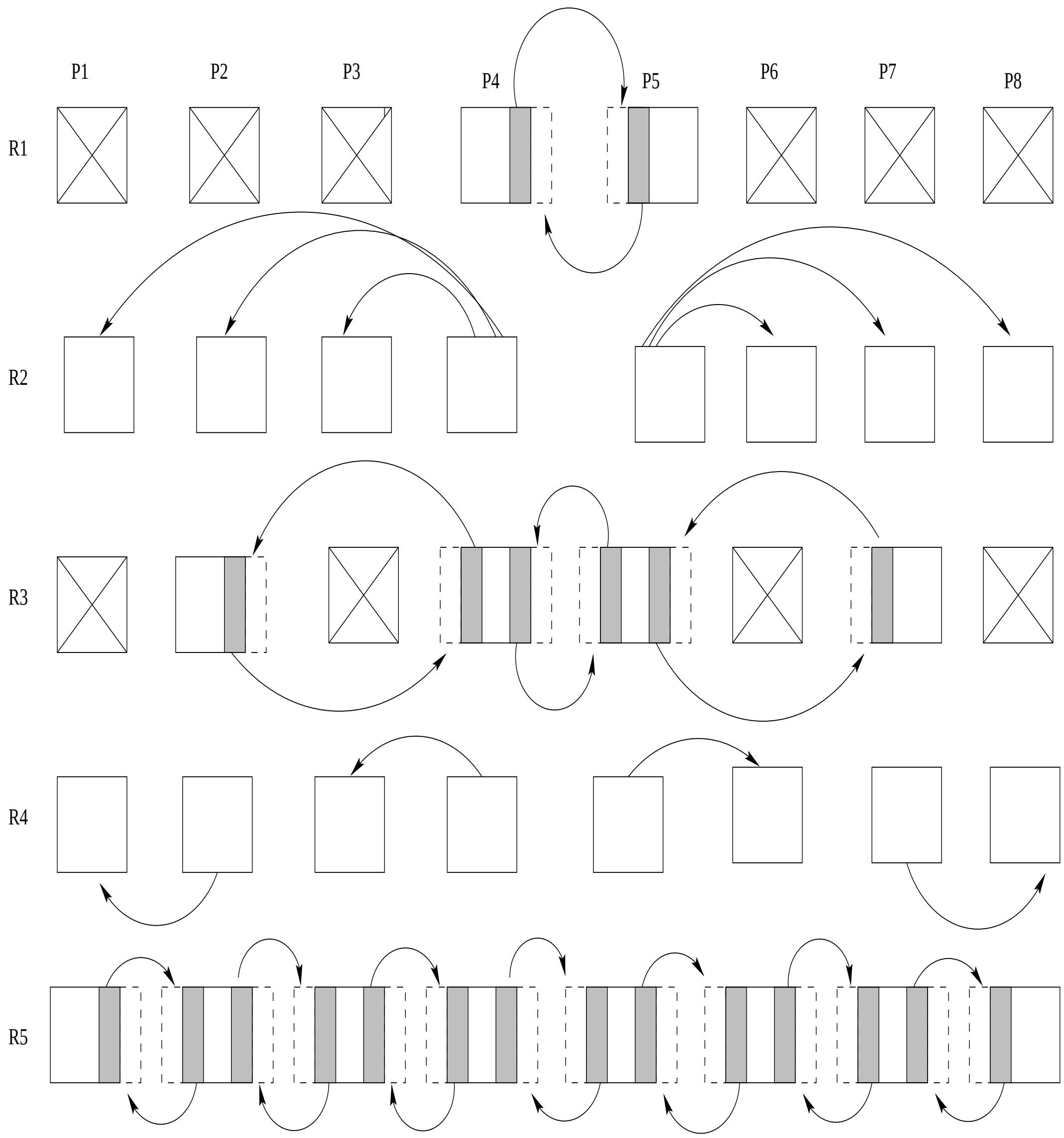

Figure 5: The transfer of data in the k direction for 8 processors united in three multigrid levels. The processors are denoted as P1 to P8. The five rows are denoted as R1 to R5 and they represent consecutive steps of the computational process. The shaded areas indicate the portion of the subdomain which is transferred to its neighbor to the area bounded by the dotted line. Idle processors are displayed by boxes which have two crossing diagonal lines.

In this example, the initial multigrid tree consists of three levels where each level contains one grid. The initial problem was decomposed into 8 subproblems described by the corresponding multigrid trees and processed on 8 concurrent processors. As shown in Fig. 5, in level 1 only 2 processors P4 and P5 in row R1 are used . Otherwise, the blocks become thinner than the size of the ENO template, which is typically equal to 3. After the completion of the iterations in level 1, processors P1, P2 and P3 are loaded by the flow-field from processor P4, and processors P6,P7,P8 are similarly loaded from processor P5 (see row R2). In the second multigrid level, 4 processors are active: P2,P4,P5 and P7, which exchange data as indicated in row R3. After the completion of the iterations in level 2, processors P1, P3,P6 and P8 are loaded by the data from processors P2,P4,P5 and P7, respectively. Finally, in the finest third level the whole set of 8 processors is activated and the information is exchanged among the processors and theirs left and right neighbours as shown in R5. In the above illustration, the information exchange was performed in direction of the decomposition. When the decomposition is applied in both i and k directions the communications scheme becomes more complicated. Still the communications matrix is sparse.

	11	12	13	14	21	22	23	24	31	32	33	34	41	42	43	44	51	52	53	54
11		k			i												i			
12	k		k			i												i		
13		k		k			i												i	
14			k					i												i
21	i					k			i											
22		i			k		k			i										
23			i			k		k			i									
24				i			k					i				i				
31					i					k			i							
32						i			k		k			i						
33							i			k		k			i					
34								i			k					i				
41									i					k			i			
42										i			k		k			i		
43											i			k		k			i	
44								i				i			k					i
51	i												i					k		
52		i												i			k		k	
53			i												i			k		k
54				i												i			k	

Figure 6: Communication between 20 (5×4) processors in the i and k directions

This is illustrated in Fig. 6 where communications scheme for 20 parallel processors is given for a decomposition 5×4 in the i and k directions, respectively. The numbering of the processors is done in the following way: 34, for example, is a two-dimensional position of the corresponding block where #3 indicates the i direction and #4 the k direction. Entries in this table contain either the letters i or k or empty which indicates the direction (if any) of the information exchange. For each two processors participating in the communication, only one direction is activated. Most entries in the table are empty indicating that no exchange of information is taking place. This shows that there is no global exchange of information. Globality of this type can substantially degrade the performance.

4 Results and Discussion

The purpose of the present runs was to speedup the serial computation and to verify the scalability of the concurrent code. Three multiprocessors have been used: SP2,MOSIX[1, 2] and Origin 2000. In MOSIX the PVM [11] software package was used while in SP2 both MPI [13] and PVM software packages have been used. On Origin 2000 only the MPI software package was used. The performance of the parallelized code for a full-size aerodynamic test-case (the high-Reynolds viscous computation for the transonic ONERA wing, described in section 2) has been checked on different multiprocessor configurations with up to 64 parallel processors.

4.1 The Multiprocessors

MOSIX is an enhancement of UNIX that allows distributed-memory multicomputers, including a LAN-connected network of workstations (NOW), to share their resources by supporting preemptive process migration and dynamic load balancing among homogeneous subsets of nodes.

The NOW MOSIX is designed to run on configurations that include personal workstations, file servers and CPU servers that are connected by LANs, a shared bus, or fast interconnection networks. In these configurations each workstation (node) is an independent computer, with one or more (shared-memory) processors, local memory, and communication and I/O devices. Users interact with the multicomputer through their own workstations.

Parallel applications can be executed on MOSIX by simply creating many processes, just like a single-machine environment. The operating system automatically attempts to optimize the process allocations and balance the load.

The system image model of MOSIX is a NOW, in which all the user's processes seem to run at the user's "home" workstation. Each new process is created at the same site (or sites) as its parent process. All the processes of each user have the execution environment of the user's home workstation. Processes that migrate to other (remote) workstations use local resources if possible, but interact with the user's environment through the user's workstation.

The NOW MOSIX is compatible with BSDI's BSD/OS Release 2.0, which is based on Release 4.4BSD-Lite from the Computer Systems Research Group at UC Berkeley. The current implementation is for Intel's X86/Pentium-based workstations and PCs.

IBM SP2 has 64 processors, 62 of which are of the type Thin Node 2; each node includes 128 Mbytes memory and a 4.5-Gbyte internal disk. The two remaining processors are of type Thin Node 2, in which each node has 512 Mbytes of memory and 4.5-Gbyte internal disk. Each processor has a POWER2 architecture RS/6000 processor running at 66.7 MHz. The processors are interconnected with a multistage switch made up of bidirectional 4×4 crossbars. The SP2 is theoretically capable of 266 Mflops and it contains the AIX Windows Environment/6000.

Origin 2000 The Origin family is both modular and scalable; that is, it can be increased in size (scaled) by adding nodes (or node boards) to the interconnection fabric. Each Node board can contain up to two R10000 processors, with accompanying cache, directory, main memory, and interfaces to both I/O devices and the interconnection fabric.

The interconnection fabric (called the CrayLink Interconnect) replaces the shared bus of the Everest architecture with a web of point-to-point links that simultaneously connect the nodes to each other and present a multitude of paths from one node to another.

The MIPS family of 64-bit processors provide a single uniform virtual address space for user processes. The Origin family uses the R10000 processor which defines a 244, 16 terabyte (TB) user-addressable virtual address space.

The Origin family has a distributed shared-memory architecture, in which shared main memory is distributed amongst the nodes. This shared memory is accessible to every processor in the system.

4.2 The Model Problem

This test-case consists of an aerodynamic problem for computing high-Reynolds viscous flow influenced by transonic shocks and their interaction with a boundary layer. The three-dimensional *ONERA M-6* wing was especially designed and wind tunnel tested in order to allow a comparison among different Navier-Stokes and Euler solvers. The serial results for the case are described in subsection 2.3.

In the serial computation, three multigrid levels were used where each level contains one grid. The coarse grid contains $44 \times 16 \times 8$ computational cells in streamwise, normal to surface and span directions, respectively. The medium grid consists of $88 \times 16 \times 16$ cells and the fine grid contains $176 \times 32 \times 32$ cells. On the third level, a local refinement is used and the actual computations cover only 24 cells in the normal direction to the surface. Due to a large number of iterations that are needed to get a convergent solution, the serial run took more than 13 hours on SP2 which is a typical representation of practical problems. Usually, the aerodynamic design demands tens of such runs (corresponding to different flight conditions) for each geometry modification. Thus, multiple processing is crucial in order to reduce substantially the execution time and to allow a rapid turnaround of the design cycle.

4.3 Results: preliminary remarks

The partition strategy proposed in subsections 3.4 and 3.5 may generate faster parallel solutions provided that the communications time is proportional to the size of transferred information. To guarantee this the length of the messages must be above the saturation phase in the graph which represents the throughput function σ versus the length of the message. Then, the communication start-up time can be ignored.

For example, the above bound varies from 16 to 32 Kbyte for both PVM and MPI software implementation on SP2. This bound is lower when implemented on Origin 2000 using MPI. In the following we are using the notation that appeared in the subsection 3.4. In our application the length of a typical message is equal to

$$\frac{N_1}{C_1}\frac{N_3}{C_3} \times \mathrm{ENO} \times \mathrm{NumVar} \times N_b$$

where N_1 and N_3 are the dimensions of the original (serial) problem in x and z directions, respectively. C_1 and C_3 are the corresponding partition factors, ENO is the size of stencil (typically 3), NumVar is the number of variables that represents each computational cell (it is equal to 6 for three-dimensional flow problems), and N_b (which is equal to 8) is the number of bytes needed to represent each variable. In the present application (see subsection 4.2) N_x=176 and $N_k = 32$ for the fine grid and the message length is approximately estimated by

$$\frac{800\mathrm{Kbytes}}{C_t}$$

where C_t is the total number of parallel processors which are used (as $C_t = C_1 \times C_3$). Thus, even 64 processors on SP2 provide an acceptable level of message passing (which is optimal for up to 32 processors). It also means that on coarser levels a certain loss of efficiency of the message passing on 25-30 processors is expected on SP2 and multiprocessors with similar saturation bounds of σ. Due to a more favorable bound for saturation no loss of efficiency is expected on Origin 2000 with up to 128 processors. (For typical large-size 3-D flow problems these figures are similar or even more favorable).

Another point to be checked before investigating the scalability of a specific problem is the performance of the concurrent code on a small number of processors (typically 4 to 8). The communications overhead for this model problem is below 25% for both PVM and MPI on SP2, about 26% on Origin 2000 with MPI and about 40% on MOSIX with PVM.

4.4 Parallel Performance

Notation that are being used in this subsection:

Serial time	$Tser$
Number of processors	N
Parallel time	$Tpar$
Speedup	$S = Tser/Tpar$
Efficiency	$E = S/N$

4.4.1 Results on SP2 with PVM

Here we performed the full Navier-Stokes test-case problem with three levels of multigrid on SP2 with up to 64 processors using the PVM message passing. The results are summarized in Table 1.

$N = C_1 \cdot C_3$	Tser (sec.)	Tpar (sec.)	Speedup	Efficiency
4=2×2	50808	16841	3.01	0.75
8=4×2	50808	7834	6.49	0.81
16=4×4	50808	4400	11.55	0.72
20=5×4	50808	3660	14.08	0.69
32=8×4	50808	2569	19.78	0.62
40=10×4	50808	2206	23.03	0.58
48=12×4	50808	1886	26.93	0.56
64=8×8	50808	1547	32.64	0.51

Table 1: Parallel performance of full Navier-Stokes equations with three levels of multigrid on SP2

Observing the results in table 1 two factors should be taken into considerations:

1. The decomposition 2×2 for 4 processors was implemented with a relatively low efficiency since the appropriate communications matrix degenerates (e.g. the processor 12 communicates with processor 22 in both i and k directions, compare to Fig. 6). This leads to the communications penalty which is heavier than in a bigger multiprocessor

configurations. The case is included as a basis for comparison with massively parallel configurations.

2. In the case when more than 8 processors participate in computation, a number of processors used on the first multigrid level is twice less than the stated number of processors. (Otherwise blocks are becoming too "thin", see the discussion in the subsection 3.5). This diminishes the expected efficiency by several percent.

Scalability of the code as demonstrated in table 1 is slightly worse than the one which is given by the optimal "square root rule" discussed in subsection 3.4. For example, when the number of processors changes from 16 to 48, the efficiency changes from 0.72 to 0.56 instead of the theoretical 0.597. (In the case of non-scalable communications this number would be equal to .46).

Table 2 summarizes the results for the same aerodynamic test case by solving the Euler problem (instead of the full Navier-Stokes problem in the previous batch of runs).

$N = C_1 \cdot C_3$	Tser (sec.)	Tpar (sec.)	Speedup	Efficiency
4=2×2	33984	14188	2.39	0.60
8=4×2	33984	6056	5.61	0.70
16=4×4	33984	3640	9.34	0.58
20=5×4	33984	3146	10.80	0.54
32=8×4	33984	2168	17.68	0.49
40=10×4	33984	1861	18.26	0.46
48=12×4	33984	1622	20.95	0.44
64=8×4	33984	1333	25.49	0.40

Table 2: Parallel performance for solving Euler equations with three levels of multigrid on SP2

In comparison between the performance of the Euler and Navier-Stokes solvers, the efficiency of the Euler solver is uniformly lower. This is due to the fact that computational task is less while the communication is the same.

4.4.2 Results on SP2 with MPI

Table 3 presents the parallel performance of the full Navier-Stokes solver on SP2.

$N = C_1 \cdot C_3$	Tser (sec.)	Tpar (sec.)	Speedup	Efficiency
4=2×2	50808	14408	3.53	0.88
16=8×2	50808	3863	13.50	0.84
16=4×4	50808	3790	13.37	0.83

Table 3: Parallel performance of the Navier-Stokes solver on SP2 with three levels of multigrid using MPI

The results show the same scalability trends as with PVM software taking into account that the maximum throughput of MPI message passing on SP2 is about 30% higher. Comparison for two differently configured 16 processors reflects the influence of the proportionality rule derived in the subsection 3.4 on the efficiency of parallelization.

4.4.3 On MOSIX with PVM

The results for several multiprocessor configurations are given in table 4.

$N = C_1 \cdot C_3$	Directions of decomposition	Speedup	Efficiency
4=2×2	i, k	2.22	0.55
8=4×2	i, k	4.81	0.60
20=5×4	i, k	10.01	0.50

Table 4: Parallel performance of the Navier-Stokes solver on MOSIX with three levels of multigrid using PVM

Compared to the results on SP2 with the same message passing software, it may be observed that the communications overhead is higher on MOSIX. The scalability behaviour is similar to that on SP2.

4.4.4 Results on Origin 2000 with MPI

Table 5 presents the parallel performance of the full Navier-Stokes solver on Origin 2000.

$N = C_1 \cdot C_3$	Tser (sec.)	Tpar (sec.)	Speedup	Efficiency
4=2×2	32700	11070	2.95	0.74
16=4×4	32700	2850	11.47	0.72
64=8×8	32700	878	37.24	0.58

Table 5: Parallel performance of the Navier-Stokes solver on Origin 2000 with three levels of multigrid using MPI

The first observation in subsection 4.4.1 stressed the fact that for a 2×2 configuration there is some additional overhead. This observation is applicable also to this multiprocessor. Thus, the real scalability on Origin 2000 is demonstrated through the comparison between the configurations that contain 16 and 64 processors. The scalability is even slightly better than the expected theoretical estimates given in subsection 3.4. The reasons for that are: 1. The sustained rate of the message-passing stays constant during the transmission even when the size of the problem is getting smaller. 2. The size of the problem fits well the size of the cache.

5 Conclusions

The domain decomposition algorithm for a full-size CFD problem was implemented on three multiprocessors: SP2, MOSIX and Origin 2000. The theoretical estimate for the overall communication time in a 2-D decomposition, denoted by W_{comm}, is estimated by $W_{comp}^{\frac{1}{2}}$ in relation to the computational work W_{comp}. This estimate was achieved on Origin 2000 and the efficiency was close to 60% on 64 processors. We expect that the achieved scalability of Origin 2000 will be maintained on larger size problems. On the other hand, the scalability level on SP2 and MOSIX was less than the theoretical estimate due to the fact that the communication rate is not sustained when the size of the problem becomes smaller. We expect better scalability on larger size problems on SP2 and MOSIX similar to what was achieved on Origin 2000.

References

[1] A. Barak, S. Guday, and R. Wheeler, "The MOSIX Distributed Operating System, Load Balancing for UNIX", Lecture Notes in Computer Science, Vol. 672, Springer-Verlag, 1993.

[2] A. Barak, O.Laden, and Z. Yarom, "The NOW MOSIX and its Preemptive Process Migration Scheme", IEEE TCOS, 7,2: 5-11, 1995.

[3] Baldwin, B.S., Lomax, H., *Thin layer Approximation and Algebraic Model for Separated Turblent flows*, AIAA paper 78-257, 1978.

[4] Brandt, A., "Multi-Level Adaptive Computations in Fluid Dynamics," AIAA Journal, Vol.18, No. 10, 1980, pp. 100-108.

[5] Brandt, A. and Yavneh, I., "On Multigrid Solution of High-Reynolds Incompressible Entering Flows", J. Comput. Phys.101 (1), 151-164 (1992).

[6] B. Epstein, A. Jacobs and A. Nachshon, " Aerodynamically Accurate Three-Dimensional Navier - Stokes Method", AIAA Journal, Vol. 35, No. 6, 1997, pp. 1089-1090.

[7] Epstein, B and Nachshon, A., "An ENO Navier-Stokes Solver Applied to 2-D Subsonic Transonic and Hypersonic Aerodynamic Flows", AIAA paper 94-0303, 1994, Reno, NV, USA.

[8] B. Epstein, A. Jacobs and A. Nachshon, "Viscous Computations by a 3-D ENO Navier-Stokes Method", Proceedings of the 37th Israel Conference on Aviation and Astronautics, February 1997, pp. 98-109.

[9] Epstein, B., Luntz, A. L., and Nachshon, A., "Multigrid Transonic Computations Arbitrary Aircraft Configurations," Journal of Aircraft, Vol. 26, No. 8, 1989, pp. 751-759.

[10] Epstein, B., Yavneh, I., A. Averbuch, "An Accurate ENO Driven Multigrid Method Applied to 3-D Turbulent Transonic Flows", Proceedings of the AIAA 14th Computational Fluid Dynamics Conference, pp. 705-715, Norfolk, Virginia, June 1999.

[11] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderaini, "PVM: parallel virtual machine", The MIT Press, Cambridge, Mass., 1994.

[12] Harten, A., Engquist, B.,Osher, S., and Chakravarthy, S., "Uniformly High Order Accurate Non-oscillatory Schemes, I," I. Comput. Phys., Vol. 71, 1987, pp.231-303.

[13] P.S. Pacheco, "Parallel Programming with MPI", Morgan Kaufmann Publishers, 1997.

[14] Radespiel, R., and Rossow, C., "An Efficient Cell-Vertex Multigrid Scheme for the Three-Dimensional Navier-Stokes Equations," AIAA paper 89-1953, 1989.

[15] Shu, C.-W., and Osher, S., "Efficient Implementation of Essentially Non-oscillatory Shock-Capturing Schemes," I. Comput. Phys., Vol. 83, 1989, pp.32-78.

[16] Turkel, E. and Vatsa, V. N., "Effect of Artificial Viscosity on Three-Dimensional Flow Solutions," AIAA Iournal, Vol.32, No.1, 1994, pp.39-45.