INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 16, 303–327 (1993)

COMPUTATION OF FLUID FLOW WITH A PARALLEL MULTIGRID SOLVER

E. SCHRECK AND M. PERIĆ

Lehrstuhl für Strömungsmechanik, Universität Erlangen-Nürnberg, Cauerstr. 4, D-8520 Erlangen, Germany

SUMMARY

A finite volume numerical method for the prediction of fluid flow and heat transfer in simple geometries was parallelized using a domain decomposition approach. The method is implicit, uses a colocated arrangement of variables and is based on the SIMPLE algorithm for pressure–velocity coupling. Discretization is based on second-order central difference approximations. The algebraic equation systems are solved by the ILU method of Stone.[1] To accelerate the convergence, a multigrid technique was used. The efficiency was examined on three different parallel computers for laminar flow in a pipe with an orifice and natural convection in a closed cavity. It is shown that the total efficiency is made up of three major factors: *numerical efficiency*, *parallel efficiency* and *load-balancing efficiency*. The first two factors were thoroughly investigated, and a model for predicting the parallel efficiency on various computers is presented. Test calculations indicate reasonable total efficiency and favourable dependence on grid size and the number of processors.

KEY WORDS Parallel computing Finite volume method Implicit method Multigrid method Domain decomposition

1. INTRODUCTION

Computational fluid dynamics consumes a large part of available computer resources. The need for numerical solutions of fluid flows and the accuracy demands are growing as optimization requirements become more stringent. Many solution methods based on finite difference, finite volume or finite element approaches have been developed. A number of commercial codes are available. The desire to solve larger problems more accurately increases the demand for efficiency.

Using vector supercomputers is nearly standard today. However, most algorithms cannot be vectorized fully. Also, the efficiency of vector processing often depends strongly on the vector length. The transport of data from memory to the vector pipes is a bottleneck which limits the computation speed. The performance of vector processors cannot be increased indefinitely since the chip miniaturization is approaching its limits.

As an alternative, parallel computers offer the promise of scaleable arithmetic performance. They can employ microprocessors which are relatively inexpensive but, nevertheless, of high performance.

Parallel computers can either be of shared memory or distributed memory architecture. Codes for shared memory are simpler to write, but memory access conflicts and the memory to CPU bottleneck prevent this architecture from being truly scaleable. For distributed-memory architectures, no such problems arise. Since each processor has its local memory, there is the possibility of parallel memory access without conflict. However, there is difficulty in writing programs for such

0271–2091/93/040303–25$17.50

Received April 1992
Revised October 1992

computers. Without communication libraries (which are now available for some parallel computers), a programmer himself has to do the load balancing and the communication between processors. This has to be done for each program, which is one reason for slow acceptance of such computers.

In computational fluid dynamics, coupled non-linear systems of partial differential equations have to be solved. Differential operators are of local character, so the solution domain can easily be subdivided into subdomains, and each subdomain can be assigned to a single processor. Communication between the processors is needed only to exchange the data at the subdomain boundaries. Therefore, one would expect high efficiency if the partitions are properly chosen. For explicit solution methods, parallelization is relatively simple; however, for implicit methods — which are important for steady flows and flows with slow transients — parallelization is less trivial. The partitioning generally decreases the convergence rate. In order to achieve efficiency, it is necessary to optimize the coupling of the subdomains.

To examine these effects, a parallel algorithm based on domain decomposition has been developed and applied to several problems on different parallel computers. The main factors influencing the performance were identified and their effects were studied by measuring computing times as a function of grid size and number of processors used. In particular, the numerical efficiency or convergence rate and the parallel efficiency, which depends on communication overhead, were studied. For the latter, a simple model equation was derived which reproduces the measured values fairly well.

2. BASIC ALGORITHM

The fluid dynamical problems considered in this study are steady, two-dimensional, laminar flows in rectangular domains. The conservation equations governing the transport of mass, momentum and heat read:

$$\operatorname{div}(\rho \mathbf{V}) = 0, \tag{1}$$

$$\operatorname{div}(\rho U_i \mathbf{V} - \mu \operatorname{grad} U_i + P\mathbf{i}_i) = \rho g_i, \tag{2}$$

$$\operatorname{div}\left(\rho T \mathbf{V} - \frac{\mu}{Pr} \operatorname{grad} T\right) = 0. \tag{3}$$

Here ρ is the density, U_i (or U, V) are the components of the velocity vector $\mathbf{V}$ in the Cartesian coordinate directions x_i, P is the pressure, T is the temperature, Pr is the Prandtl number, μ is the dynamic viscosity, g_i is the component of the gravitational acceleration vector and $\mathbf{i}_i$ is the unit vector in the x_i-direction. Fluid properties are assumed constant, except in the buoyancy term (Boussinesq approximation).

The solution domain is discretized into rectangular cells. The transport equations are applied to the finite control volumes (CVs), leading to balance equations involving the fluxes through the CV faces, F_i, where $i = \mathrm{e}, \mathrm{w}, \mathrm{n}, \mathrm{s}$ (see Figure 1) and the volumetric sources, Q, for each CV:

$$F_\mathrm{e} + F_\mathrm{n} + F_\mathrm{w} + F_\mathrm{s} = Q. \tag{4}$$

The mass fluxes through CV faces satisfy the continuity equation and are used to compute the convective fluxes in the other transport equations in the next iteration. This is the simplest way of linearizing the convective terms. For example, convective flux of ϕ, where ϕ stands for U, V or T, through CV face e may be expressed as:

$$F_\mathrm{e}^\mathrm{c} = \dot{m}_\mathrm{e} \phi_\mathrm{e}. \tag{5}$$

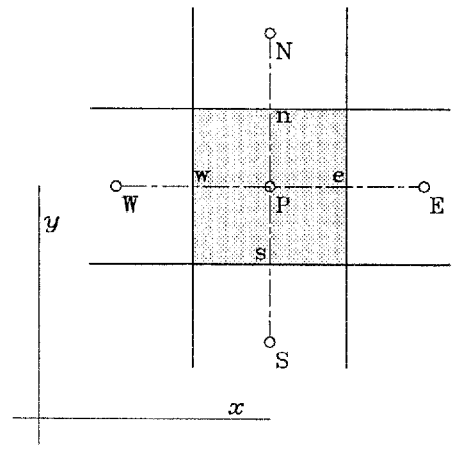

Figure 1. A typical control volume and labelling scheme

Here $\dot{m}_e$ is the mass flux through the CV face and ϕ_e is assumed to represent the mean value of ϕ over the CV face; it is expressed in terms of nodal values using second-order central difference approximations. The diffusive contribution to the flux F is

$$F_e^d = \left(\Gamma_\phi \frac{\partial \phi}{\partial x}\right)_e \Delta y \approx \frac{\Gamma_{\phi,e}\Delta y}{x_E - x_P}(\phi_E - \phi_P), \quad \text{with} \quad \Gamma_\phi = \mu \quad \text{or} \quad \frac{\mu}{Pr} \tag{6}$$

which also represents a central difference approximation. The source terms are approximated by assuming that the volumetric source at CV centre, P, represents the mean value over the whole CV.

For each CV the above approach leads to an algebraic equation of the form

$$A_P \phi_P + \sum_{nb} A_{nb}\phi_{nb} = Q_P, \tag{7}$$

where index nb runs over the nearest neighbour CV centres E, W, N and S. For the whole solution domain, a matrix equation

$$[A]\{\phi\} = \{Q\} \tag{8}$$

results. $[A]$ is the square coefficient matrix which, for the above discretization scheme and structured grids, has non-zero elements on five diagonals only. $\{\phi\}$ and $\{Q\}$ are column matrices whose elements are nodal values of the unknown ϕ and the source term of equation (7), arranged sequentially along grid lines.

The linearized equations are relaxed iteratively using an iteration matrix $[M]$ as follows:

$$[M]\{\phi^m\} = \{Q\} - [A - M]\{\phi^{m-1}\}, \tag{9}$$

where m is the iteration counter. These iterations are called *inner* iterations. In the present study the incomplete lower–upper decomposition (ILU) solver of Stone[1] is used. It is similar to the standard ILU method in that the matrix $[M]$ is a product of a lower $[L]$ and an upper $[U]$ triangular matrix, which have the same sparcity as the matrix $[A]$. However, Stone's method uses the smoothness property of the solutions of partial differential equations to introduce approximations which minimize the product $[A - M]\{\phi\}$, thus leading to a much faster convergence than the standard ILU method. The solver employs a relaxation parameter α, whose value is chosen in

the range $0 \leqslant \alpha < 1$. Optimum value is problem-dependent, but usually good results are obtained with $\alpha = 0{\cdot}92$, which is the value used in the present computations. For $\alpha = 0$, the method reduces to the standard ILU solver.

The coupled set of non-linear equations for U, V, T and P is solved sequentially using SIMPLE algorithm[2] for pressure–velocity coupling. The discretized momentum equations are assembled using the latest available values for the other variables (pressure and mass fluxes) and solved with the ILU solver. For a single domain, one inner iteration is sufficient. The mass fluxes calculated from velocity components so obtained do not satisfy the continuity equation. Mass conservation is enforced by correcting the velocities by adding a pressure gradient correction; the correction is

$$U'_P = -\frac{\Delta y}{A_P^u}(P'_e - P'_w), \tag{10}$$

where A_P^u is the central coefficient of the U-equation. Inserting these velocity corrections into descretized continuity equation leads to a pressure-correction equation of Poisson type. The source term in the pressure-correction equation is the imbalance in the uncorrected mass fluxes. This equation is also solved with the ILU solver; typically, six to ten inner iterations are required. The mass fluxes, velocities, and pressure are then corrected using the pressure correction. For natural convection the energy equation is also solved. It is coupled to the momentum equations

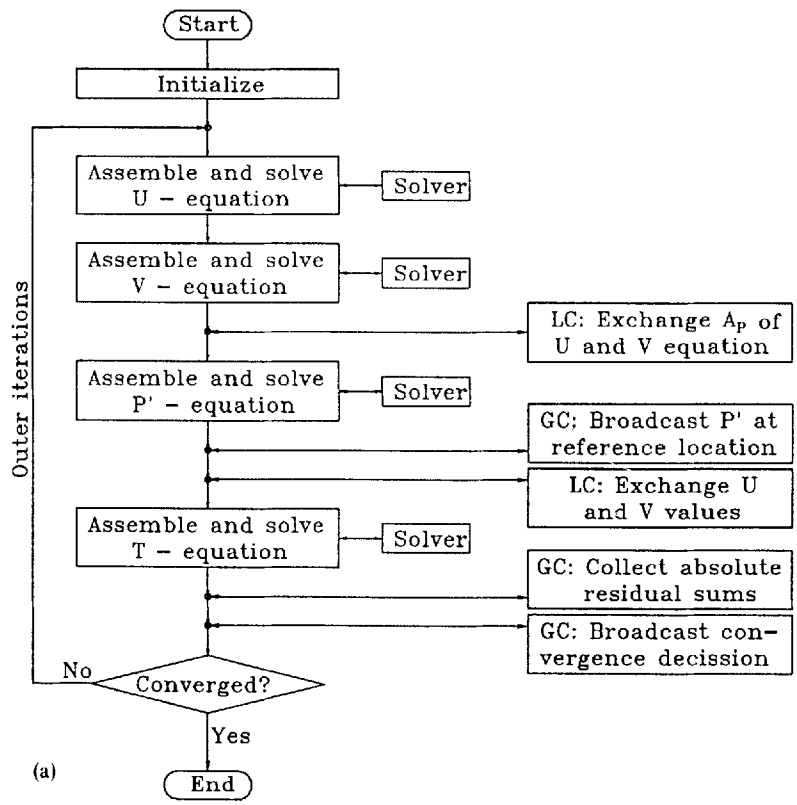

Figure 2. Flow chart of the outer iteration loop (a) and the inner iteration loop (b), also showing local (LC) and global (GC) communication in the EI mode

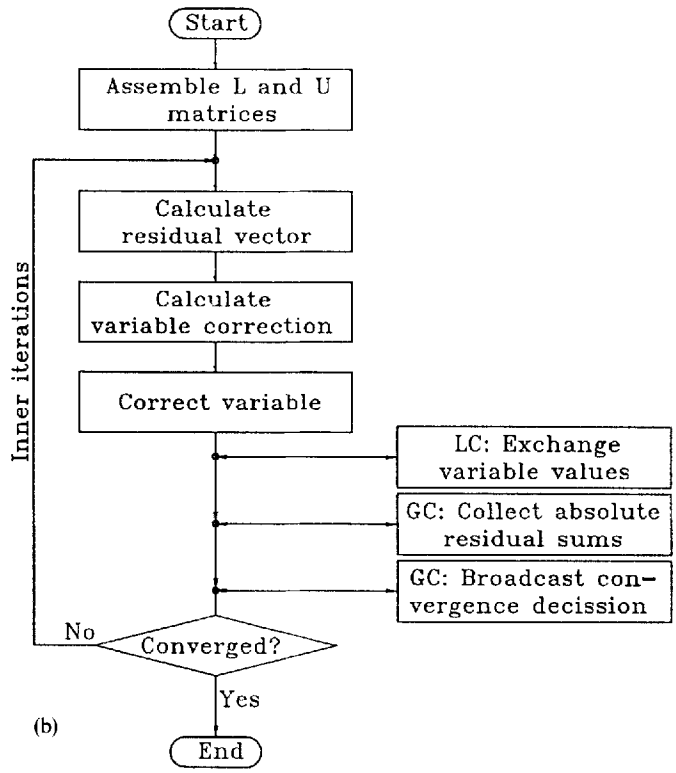

Figure 2. (Continued)

through the buoyancy term in the latter and the velocities in the former. This completes one *outer* iteration of the SIMPLE procedure. The coefficients of the difference equations are then updated and the procedure repeated until convergence is achieved. The convergence criterion requires that the sum of absolute residuals in each equation be reduced by a prescribed amount (usually four orders of magnitude). A flow chart of the outer and inner iteration procedure is presented in Figure 2. More details of the method can be found in References 3 and 4.

This procedure efficiently removes only those components of the error whose wavelengths are comparable to the grid spacing. For this reason, the number of outer iterations increases linearly with the number of grid points, resulting in a quadratic increase in computing time. To accelerate convergence, a multigrid (MG) scheme was implemented. A solution is first obtained on the coarsest grid using the strategy described above. This solution is then interpolated to obtain a starting iterate on the next finer grid (the so-called 'full multigrid procedure'). After performing a few (two to five) *outer* iterations on the finer grid, the process is transferred to the coarser grid. The equations solved on the coarse grid are those solved in the first step, when this was the only grid used, except for additional source terms.[4] This solution yields a correction to the fine-grid solution, which removes error components with long wavelength. The procedure is repeated until the solution on the finest grid converges using the so called 'V-cycles'.[4]

The coarser CVs are constructed by amalgamating four fine-grid CVs, see Figure 3. The nodes on the two grids do not coincide and, therefore, transfer of variables between the two grids has to

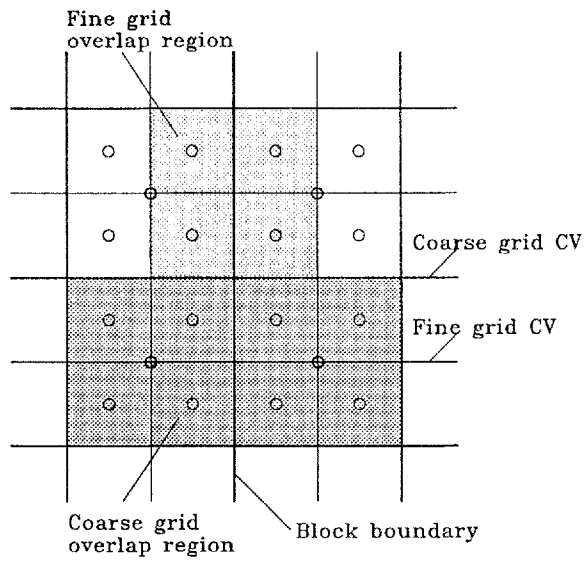

Figure 3. Coarse- and fine-grid CVs of the multigrid algorithm, also showing overlap region near block boundary

be performed by interpolation. With the MG method, the computing time increases linearly as the grid is refined, resulting in substantial savings.

3. PARALLELIZATION STRATEGY

In this section, the strategy for the concurrent algorithm is described. The method is based on data parallelism, i.e. the same program runs on every processor with different data. The solution domain is subdivided into non-overlapping subdomains, and each subdomain is assigned to one processor. As can be seen from equation (7), the equation for each CV requires values from its neighbours. Therefore, the control volumes along subdomain boundaries need values from CVs allocated to neighbouring processors. In a multiprocessor with distributed memory, it is necessary for each processor to store some data calculated by neighbouring processors in its memory. Each time a processor updates a variable which is needed by the neighbour processor, it is copied to the neighbour processor's memory. We assume here that a five-point discretization scheme is used so that data from only one node on the other side of the subdomain boundary is needed; see Figure 4. For more complex schemes, data from more than one line of CVs along the boundary has to be exchanged and stored. This region is called the 'overlap region'.

In this study, only structured regular grids are considered. Each subdomain is, therefore, a rectangle of $N_i \times N_j$ CVs. Also, each subdomain boundary is assumed to be common to only two processors; this condition has to be relaxed in complex geometries, where the global (overall) grid may be unstructured or block-structured.

In the SIMPLE algorithm, the variable values needed to calculate the coefficients and source terms are taken from the previous iteration. Therefore, they can be calculated in parallel.

Due to recursive data dependencies in the ILU algorithm, its global parallelization so that convergence is achieved in the same number of iterations as on a single processor is possible only when the dimension of the processor configuration is one order lower than the grid dimension[5] (a ring for 2D and an array for 3D problems). The same is true for conjugate gradient type of

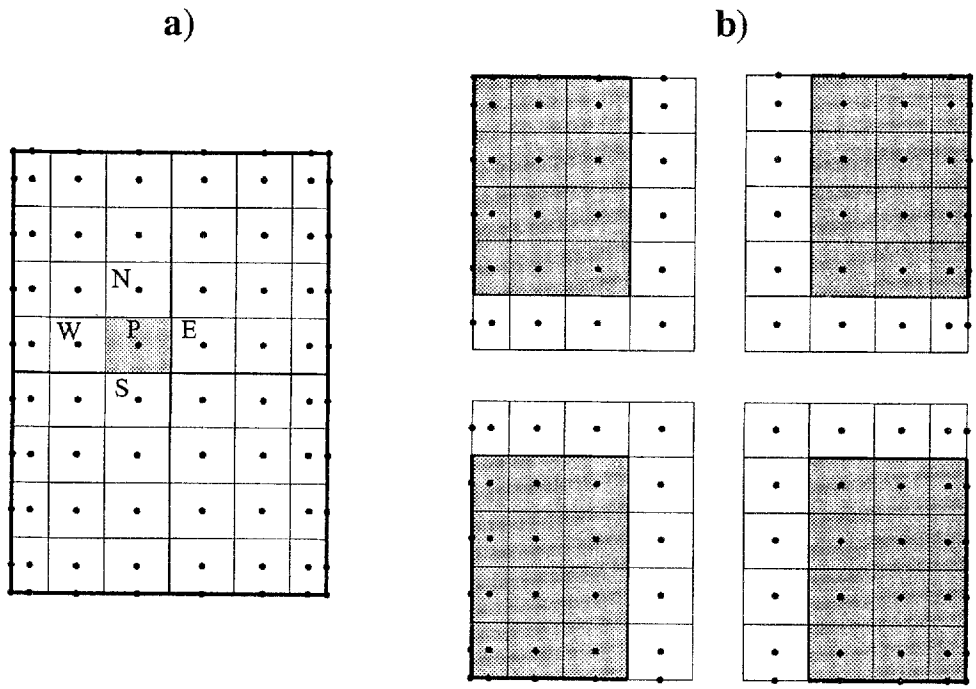

Figure 4. Decomposition of a single domain (a) into four subdomains (b), showing the overlap region (open cells)

solvers, which are also popular in computational fluid dynamics. The loss of time due to communication and synchronization is strongly dependent on the number of grid points per processor and is a limiting factor for complex grids, since such a parallelization is only possible on logically rectangular blocks. Therefore, the equation system is split into subsystems, one for each subdomain, and these smaller systems are relaxed separately. Of course, this decreases the convergence rate, but it offers more flexibility and, in most cases, yields a shorter computing time than global parallelization of the single domain solver. There are two possible modes of communication: data can be exchanged after each inner iteration (EI mode, shown in Figure 2) or only after a complete outer iteration (EO mode). One expects the EO mode to have slower convergence than the EI mode due to weaker coupling between the subdomains; however, the communication is substantially reduced, so the overall cost may be lower for computers with long set-up times; see below.

The exchange of the data between the neighbouring processors is *local* and can be performed in parallel. In the EI mode, it is done after each inner iteration, before assembling the pressure-correction equation and after correcting velocity field, as shown in Figure 2. In EO mode, local communication in solver is done only after the final inner iteration. In the multigrid case, additional data exchange is needed after the restriction of variable values from the fine to the coarse grid, since the neighbour coarse grid nodes lie outside the overlap region of the finer grid; see Figure 3.

Some *global* communication is also needed, e.g., to sum the residuals for completion criterion. The residual sums of all subdomains have to be collected, and the decision whether to stop or go has to be broadcast to all the processors. This is normally done after each outer iteration, and—unless a fixed number of inner iterations is prescribed—after each inner iteration. In the

present algorithm, pressure is kept fixed at one node, but pressure correction is allowed to float; therefore, the pressure correction value at the reference node has to be substracted from values at all other nodes. This requires broadcasting the reference pressure correction, which is done once per outer iteration; see Figure 2.

In the above discussion, the term 'neighbouring processor' means a processor which performs calculations on a neighbouring subdomain, i.e. a logical neighbour processor. If the logical neighbouring processors are connected via hardware channels, the communication may be very fast. On the other hand, the communication with physically remote processors is slower and may have long set-up times. However, if processors communicate via a bus system, there is no distinction between neighbour and remote processors.

One objective of the present study is to obtain portable parallel programs, i.e. it should be possible to implement them easily on parallel computers with different architectures. This aim is achieved by separating the computation and communication into different subroutines. The communication subroutines are further divided into communication primitives and higher-level routines. When porting the program to another computer, only the communication primitives, which are small subroutines, have to be rewritten. It is expected that in the future these routines will be available in libraries so that porting of parallel CFD programs will become akin to porting of programs for graphical data presentation.

4. EFFICIENCY ANALYSIS

For analysis of the performance of parallel algorithms and comparison of algorithms and parallel computers, the speed-up factor and efficiency are the commonly used measures.[6] They are defined as follows:

$$S_n = \frac{T_s}{T_n}, \qquad E_n^{\text{tot}} = \frac{T_s}{nT_n},$$

where T_s is the execution time for the best serial algorithm and T_n is the execution time for parallel algorithm using n processors.

The achieved speed-up is typically less than n (the ideal case), which corresponds to an efficiency of 100%. Note that the efficiency for $n=1$ may not be 100%, as the parallelized algorithm may be slower on one processor than the best serial algorithm; this is an important issue which is often ignored by using T_1 instead of T_s in the above expressions. The loss of efficiency is mainly due to the following factors:

(1) time needed for local and global communication which halts computation (*parallel efficiency*, E_n^{par}),
(2) increase in the number of inner and/or outer iterations necessary to fulfil the convergence criterion, due to the changes in the algorithm required to parallelize it (*numerical efficiency*, E_n^{num}),
(3) idle time of processors caused by uneven load, i.e. different number of CVs per processor (*load-balancing efficiency*, E_n^{lb}).

If the processors are synchronized to start each iteration at the same time, the duration of an iteration is dictated by the processor with the largest number of CVs; all other processors have some idle periods (delays may also occur due to boundary conditions, different number of neighbours, etc., but these effects are neglected in this study). This effect may be avoided by making sure that all subdomains have the same number of CVs. Under these conditions $E_n^{\text{lb}}=1$, so only the first two factors need be considered. This is done in the present study.

The total execution time of a parallel algorithm on n processors is the sum of calculation time, T_n^{calc}, and communication time, T_n^{com}:

$$T_n = T_n^{\text{calc}} + T_n^{\text{com}} = N_n^{\text{cv}} \tau i_n k_n + t_n^{\text{com}} k_n, \tag{11}$$

where N_n^{cv} is the maximum number of CVs treated by any processor, τ is the computing time per floating point operation, i_n is the mean number of floating point operations per outer iteration per CV, k_n is the number of outer iterations and t_n^{com} is the mean communication time per outer iteration.* Values with subscript n depend on the number of processors used.

Inserting expression (11) into the definition of the total efficiency yields

$$E_n^{\text{tot}} = \frac{N^{\text{cv}} \tau i_s k_s}{n(N_n^{\text{cv}} \tau i_n k_n + t_n^{\text{com}} k_n)} = \frac{i_s k_s}{i_n k_n} \frac{N^{\text{cv}}}{n N_n^{\text{cv}}} \frac{1}{1 + t_n^{\text{com}}/t_n^{\text{calc}}} = E_n^{\text{num}} E_n^{\text{lb}} E_n^{\text{par}}, \tag{12}$$

where $t_n^{\text{calc}} = N_n^{\text{cv}} \tau i_n$ is the mean calculation time per outer iteration and N^{cv} is the total number of CVs. Since, in the present study, all subdomains always had the same number of CVs, $N_n^{\text{cv}} = N^{\text{cv}}/n$ and $E_n^{\text{lb}} = 1$, so the total efficiency equals the product of the numerical and parallel efficiencies.

The numerical efficiency is defined as the ratio of the total number of floating-point operations per CV in the serial algorithm, $i_s k_s$, to the total number of operations in the parallel algorithm on n processors, $i_n k_n$, required to reach the same convergence criterion. It does not depend on the performance characteristics of the computer.

The parallel efficiency is defined as the ratio of the computing time when using n processors, $T_n^{\text{calc}} = N_n^{\text{cv}} \tau i_n k_n$, to the sum of computing time T_n^{calc} and communication time $t_n^{\text{com}} k_n$. The communication time can be further split into local and global communication, as will be discussed later.

In order to parallelize a numerical solution procedure, it may be necessary to modify the serial algorithm. In that case, i_1 is not the same as i_s. The number of floating-point operations per outer iteration is not constant because the number of inner iterations may vary, unless it is fixed and no convergence check is applied to the inner iterations (which is possible if one has experience with previous calculations of similar problems). In any case, the variation is usually not very large. The numerical efficiency is, therefore, not easy to measure but, by assuming that i_1 and i_n are approximately equal, the ratio of the numbers of outer iterations is a good estimate. A more exact value can be obtained by measuring the total and parallel efficiencies and calculating E_n^{num} from equation (12).

The total efficiency is easily determined by measuring the computing time necessary to reach a converged solution. The parallel efficiency cannot be measured exactly as it depends on the number of data transfers between processors, which depends on the number of inner iterations per outer iteration. However, by using a fixed number of outer iterations on one and n processors, the measured total efficiency will be equal to the parallel efficiency, since in that case $i_1 = i_n$ and $E_n^{\text{num}} = 1$. This approach was used in all calculations presented in the next section.

5. RESULTS OF TEST CALCULATIONS

The parallelized code was implemented on three different parallel computers, whose characteristics are summarized in Table I.

---

* This assumes that communication and processing cannot be simultaneous, which is not true for all parallel computers. However, it is true for computers used and, since it represents the limiting (worst) case, it is worth studying such a case first. For machines with parallelism in communication and calculation, t_n^{com} represents only the communication which halts computation.

Table I. Performance characteristics of computers used

Computer	t^{st} (μs)	R_{tr} (MB/s)	$1/\tau$ (MFlops)	t^{st}/τ
Meiko (*channels*)	22	1·4	0·45	10
Meiko (*transports*)	180	1·4	0·45	81
Parsytec (*dumb links*)	56	1·3	0·35	20
Parsytec (*message ports*)	180	1·3	0·35	63
Parsytec (*helios*)	1340	1·4	0·35	469
Suprenum[a] (*syncr.*)	2000	11·6	0·11	220
Suprenum[a] (*asyncr.*)	3310	2·0	0·11	364

[a] The set-up times for Suprenum are halved for messages of < 50 bytes.

The first was a Meiko Computing Surface with 64 T800 transputers with a clock rate of 25 MHz. Each transputer has 4 MB of memory. The four transputer links are connected to routing chips which can be programmed to establish the desired configuration. The constraints are that every transputer can be connected to at most four physical neighbours, and one transputer is connected to the host. The configurations used were the ring and the surface of a cylinder (thorus). Two communication possibilities existed: (i) the four hardwired links (*channels*), with very short set-up time, but communication only with the four nearest neighbours; (ii) *transports* – a soft link that can be established at run time to any processor, but with an order of magnitude longer set-up time (see Table I).

The second computer was a Parsytec Supercluster, which uses the same transputers (256 of them) and a similar architecture. Here, three communication possibilities existed: (i) *dumb links*, similar to the above-mentioned channels but equipped with a timeout mechanism and, therefore, somewhat slower, (ii) *message ports* with more software support and flexibility but still slower, and (iii) input/output procedures of the Helios operating system, the most comfortable but the slowest option (see Table I).

The third computer used was a Suprenum with 20 nodes divided into two clusters. Each node consists of a Motorola 68020 (20 MHz) processor, 8 MB memory, a 68881 scalar coprocessor and a Weitek vector coprocessor. Due to problems with the autovectorizing compiler, only the scalar coprocessor was used. The nodes in each cluster are connected via an intracluster-bus, offering two communication possibilities: (i) asynchronous and (ii) synchronous, with half the set-up time.

The major parameters characterizing these computers and influencing the performance of a parallel algorithm are: (i) the set-up time, t^{st}, required to enable message passing; (ii) the time needed to perform one floating-point operation, τ and (iii) the rate at which data is transferred between processors, R_{tr}. As will be shown later, the ratio of t^{st} to τ affects strongly the efficiency of parallel computing.

Test case 1

The first test case was laminar flow in a pipe with an obstacle. The Reynolds number was ($Re =$) 100, and the boundary conditions were: no slip at the walls, a parabolic velocity profile at the inlet, the radial velocity and radial gradient of the axial velocity are zero at the axis; and at the outlet, zero gradients. The grid and the predicted streamlines are shown in Figure 5. The solution domain was divided into stripes as indicated in Figure 5(a), due to the large aspect ratio of the domain (25:1). Up to 256×64 CV and 16 processors were used.

The measured efficiencies for various grid sizes and number of processors are shown in Tables II–IX. Presented are the total efficiency, E_n^{tot}, and the parallel efficiency, E_n^{par}, defined in Section 4. The numerical efficiency is the ratio of the total and parallel efficiencies.

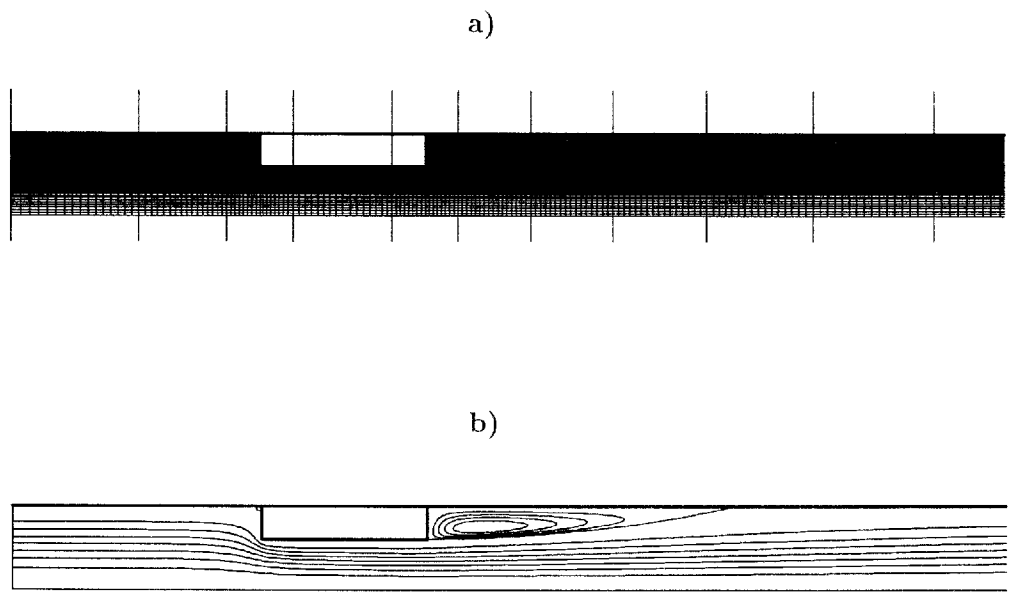

Figure 5. Numerical grid (a) and predicted streamlines (b) for the test case 1, indicating partitioning into stripes

Both single-grid (SG) and multigrid versions of the solution method were tested.

The parallel efficiencies of the single-grid algorithm with the EI mode of communication on the Meiko transputer system are shown in Table II. For a fixed number of processors, the efficiency increases as the grid is refined. At fixed grid size, the efficiency decreases as the number of processors is increased. This behaviour was observed in all cases and on all computers and is due to the following factors. When the number of CVs in each direction is increased by a factor of two, the calculation time of each processor increases by a factor of four, but the number of boundary CVs and, therefore, the communication time increase by a factor of two (ignoring set-up time). Thus the calculation time varies linearly with the number of CVs while the communication time varies as the square root of the number of CVs. Therefore, the ratio of the communication to calculation time is reduced as the grid is refined and the efficiency is increased.

In Table III, the total efficiencies are presented. They show the same dependence on the grid size and the number of processors, but are lower due to the numerical efficiency. Comparison of

Table II. Parallel efficiency for various grid sizes and numbers of processors on the Meiko transputer system (single grid, EI mode)

	E_n^{par} (%)			
n	32×8	64×16	128×32	256×64
1	100	100	100	100
2	97	99	99	99
4	89	95	98	98
8	77	90	95	97
16	57	78	90	94

Table III. Total efficiency for various grid sizes and numbers of processors on the Meiko transputer system (single grid, EI mode)

	E_n^{tot} (%)			
n	32×8	64×16	128×32	256×64
1	100	100	100	100
2	93	98	104	99
4	83	84	97	96
8	58	79	94	95
16	36	60	89	90

Table IV. Total efficiency for various grid sizes and numbers of processors on the Meiko transputer system (single grid, EO mode)

	E_n^{tot} (%)			
n	32×8	64×16	128×32	256×64
1	100	100	100	100
2	97	100	99	99
4	59	65	76	86
8	36	83	74	90
16	32	68	57	72

Tables II and III reveals that the numerical efficiency behaves like the parallel efficiency. This is because the rate of convergence is mostly affected by the ratio of the number of inner boundary nodes to the total number of nodes. For a given number of processors, the rate of convergence is less affected on finer grids. The same is true if the number of processors is reduced at constant grid size.

The effect of the introduction of inner boundaries on the rate of convergence (i.e. the numerical efficiency) cannot be predicted due to the strong non-linearity and coupling of the equations solved, it is also problem-dependent. However, the trend observed above is rather a rule than an exception, as the results of test case 2 will show. This is an important issue which is often ignored by concentrating on parallel efficiency alone.

In Table IV the total efficiencies obtained with the EO mode of communication are presented. For the EO mode, the parallel efficiency is much better due to reduced communication, but the numerical efficiency is lower due to weaker coupling between the subdomains. The ratio of the numerical efficiencies of the EI and EO modes is independent of computer performance. However, as indicated in the previous section, the parallel efficiency depends strongly on the ratio of the communication to the arithmetic performance. Therefore, on computers with slow communication, the increased E_n^{par} in the EO mode can compensate for the reduced E_n^{num}, so the EO mode might be preferable. However, on the transputer system, the EI mode is obviously the better choice as revealed by Tables III and IV. This example demonstrates the fact that the most efficient algorithm for a given problem depends on the hardware used.

Tables V and VI show the results obtained on the Suprenum. The Suprenum has a lower ratio of communication speed to arithmetic performance than the Meiko transputer system (due to the

Table V. Total efficiency for various grid sizes and numbers of processors on the Suprenum (single grid, EI mode, asynchronous)

	E_n^{tot} (%)			
n	32×8	64×16	128×32	256×64
1	100	100	100	100
2	74	91	102	96
4	40	65	88	94
8	16	45	75	90
16	5	20	51	79

Table VI. Total efficiency for various grid sizes and numbers of processors on the Suprenum (single grid, EO mode, asynchronous)

	E_n^{tot} (%)			
n	32×8	64×16	128×32	256×64
1	100	100	100	100
2	90	98	98	99
4	46	61	75	86
8	20	71	71	90
16	10	45	52	71

Table VII. Total efficiency for various grid sizes and numbers of processors on the Suprenum (single grid, EI mode, synchronous)

	E_n^{tot} (%)			
n	32×8	64×16	128×32	256×64
1	100	100	100	100
2	80	94	102	97
4	51	72	91	94
8	23	56	81	91
16	9	29	63	83

long set-up time). All trends of efficiency versus grid size and number of processors observed for the transputer system remain valid for the Suprenum. The total efficiency on the Suprenum is, except for large numbers of processors, higher for the EO communication mode than for the EI mode, since the gain in parallel efficiency in the EO mode compensates for the loss of numerical efficiency. For the transputer system, the EO mode is significantly less efficient than the EI mode, since the loss in the numerical efficiency is far greater than the gain in parallel efficiency. Table VII shows results obtained on the Suprenum using synchronous communication in the EI mode. In this case, the set-up time is half of the prior one so the efficiencies are better than those in Table V.

Table VIII. Performance of the single-grid and multigrid method on a 256 × 64 CV grid on the Meiko transputer system (EI mode)

	Single-grid method			Multigrid method		
n	Time (s)	E_n^{tot} (%)	No. of iterations	Time (s)	E_n^{tot} (%)	No. of iterations
1	27000[a]	100	1596	1390[a]	100	49
2	13690	99	1600	----		----
4	6991	96	1623	345	99	49
8	3568	95	1637	183	95	49
16	1870	90	1674	101	86	49

[a] Estimated.

The fact that the total efficiencies for the 128 × 32 CV grid and EI communication mode are higher than 100% when two processors are used is surprising. Only for this case is the number of iterations required to reach convergence lower than with one processor, which resulted in unexpected rise in efficiency. A possible explanation is that the increased number of inner iterations (with one processor, only one inner iteration is performed in momentum equations) caused the reduction in the number of outer iterations. This is an exception; in most applications, efficiency is reduced as the number of processors is increased.

The calculations discussed so far were performed with the SG algorithm. On fine grids, the multigrid method reduces the number of iterations and thus the computing time. In a multigrid algorithm, a sequence of grids or varying refinement is used. This obviously effects the efficiency of the parallelization. When a large number of processors is used on a coarse grid, the communication time outweighs the computing time (i.e. $E_n^{par} < 50\%$), see Table VIII. On the other hand, in the multigrid procedure, it is essential that a sufficient number of iterations especially on the coarsest grids be performed. Thus, for a given grid, the total efficiency will be lower for the multigrid than for the single-grid algorithm. The results of calculations presented in Table VIII demonstrate that this is indeed so.

An interesting observation from Table VIII is that the number of iterations on the finest grid in the multigrid algorithm is independent of the number of processors. This indicates that the numerical efficiency is very high. Of course, more work is done as the number of processors is increased, since the numbers of outer iterations on the coarse grids and inner iterations are increased. That the efficiency of the multigrid acceleration of convergence does not deteriorate more is due to the nature of the error components eliminated on each grid. The high-frequency errors, which are eliminated on fine grids, are local in character. Therefore, there is no need for strong coupling of subdomains on the fine grids. The low-frequency errors have global character and require treatment of the solution domain as a whole. However, these errors are eliminated on the coarsest grids, where strong coupling of subdomains is provided by the grid sparsity (larger overlap region, see Figure 3). Increasing the number of outer and inner iterations on the coarsest grids ensures that an accurate solution is obtained at a moderate increase of computing time, since one coarse-grid iteration consumes only a fraction of the time needed for one fine-grid iteration (e.g. with five-grid levels, 256 iterations on the coarsest grid last as long as one iteration on the finest grid).

The negative effect of more iterations on the coarsest grid depends on the communication performance of the computer. Especially critical is the set-up time for the initialization of data exchange, since the amount of data to be exchanged is low but the frequency is high. Of the

Table IX. Efficiency of parallel computation for test case 1 on different computers using 16 processors, different grids and different communication options

		E_n^{par} (%), SG				E_n^{tot} (%), MG
Computer	t^{st}/τ	32×8	64×16	128×32	256×64	256×64
Meiko (*channels*)	10	57	78	90	94	86
Parsytec (*dumb links*)	20	43	71	88	93	80
Parsytec (*mess. ports*)	63	29	59	82	91	72
Meiko (*transports*)	81	20	48	76	89	64
Suprenum (*synchronous*)	200	14	39	70	89	56
Suprenum (*asynchronous*)	364	8	26	57	82	42
Parsytec (*helios*)	469	4	13	37	64	24

computers used, Suprenum was the worst in this regard, as shown in Table IX; it is better to do the coarse-grid calculations on fewer processors and leave the others idle.[7] However, this approach was not attempted here. Although the multigrid method operates with lower total efficiency (86% versus 90%), it is still about 20 times faster on the 256×64 CV grid than the single-grid version, as the computing times shown in Table VIII demonstrate.

The set-up time is the crucial parameter influencing communication when the amount of transferred data is low. Table IX shows efficiencies E_{16}^{par} and E_{16}^{tot} for various grids, computers and communication options along with the ratio of t^{st} to τ. The highest efficiencies are always achieved for the shortest set-up time. The difference would diminish if the grid were further refined. The effect remains serious for the multigrid algorithm, since it always uses very coarse grids; see Table IX.

Test case 2

As the second test case, natural convection in a closed cavity is considered. The predicted isotherms and streamlines are presented in Figure 6. The direction of gravity is downward. The left and right walls were kept at constant dimensionless temperatures $T_H = 1$ and $T_C = 0$, respectively. The top and bottom walls are adiabatic. The fluid properties were chosen such that the Rayleigh number is 10^4, with Prandtl number $Pr = 0{\cdot}71$ (air). Since the geometry of this test case is square, various subdivisions were considered: n stripes in either direction or $n_x \times n_y$ blocks in x- and y-direction (where $n = n_x n_y$). This offered the posibility of studying the effects on efficiency of the shape of subdomains and the number of neighbours.

When measuring parallel efficiency, the numbers of inner iterations for the various variables were specified as follows: three for U and V, 14 for pressure correction and 4 for temperature. This choice is based on average numbers resulting from convergence criterion for inner iterations for medium-size grids. The global communication for convergence check was performed after every inner iteration.

In the first test case, the mass flow rate was prescribed and the flow had—in spite of the two recirculating regions a predominant direction from inlet to outlet. The buoyancy-driven flow is purely recirculating and the flow rate is not known *a priori*. There is an additional equation to be solved (for temperature) and the non-linearity is more pronounced.

Whereas it was natural to create subdomains by cutting the domain into stripes in the first test case, in the second test case the choice is not obvious. In order to test the effect of the type of domain decomposition on efficiency, calculations were performed on a 128×128 CV grid using

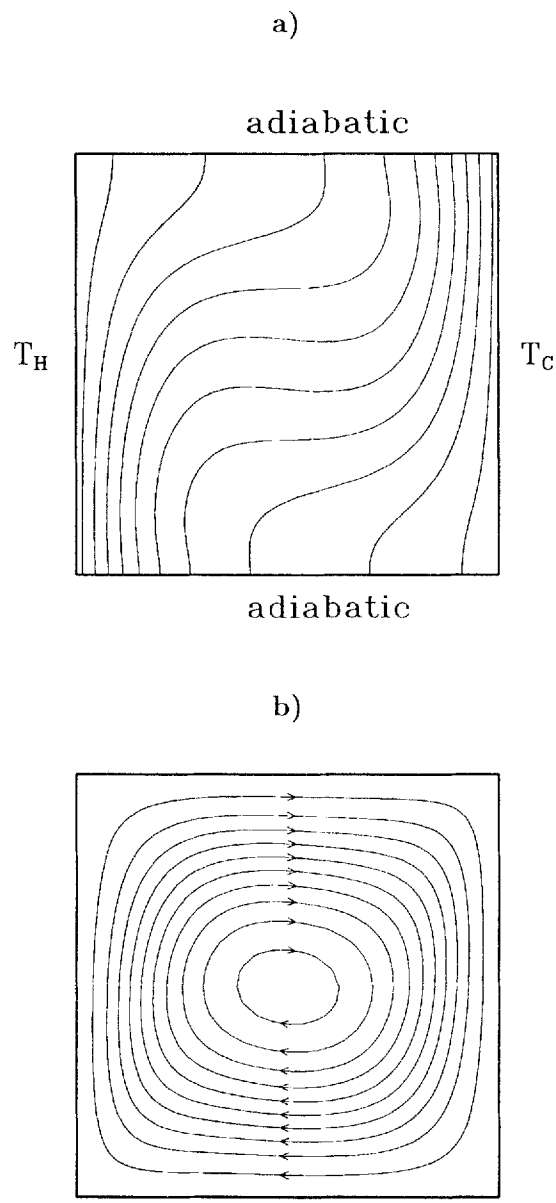

Figure 6. Predicted streamlines (a) and isotherms (b) for the test case 2

the Meiko tranputer system with various subdomain shapes, from horizontal stripes to squares to vertical stripes. The efficiencies obtained are presented in Tables X and XI. Two important effects are noticeable:

(1) the highest efficiency is obtained for square subdomains, and
(2) the efficiency is not the same for horizontal and vertical stripes of the same size.

The first effect is due to two factors. Firstly, the number of boundary values which need to be

Table X. Total efficiency for the test case 2 on the Meiko transputer system using various grids and numbers of processors (single grid, EI mode)

Processor configuration			E_n^{tot} (%)			
n	n_x	n_y	16×16	32×32	64×64	128×128
1	1	1	100	100	100	100
2	1	2	88	93	93	–
	2	1	94	96	95	–
	1	4	68	75	85	91
4	2	2	80	88	90	94
	4	1	76	79	87	91
	1	8	44	58	78	87
8	2	4	57	69	84	91
	4	2	57	71	85	90
	8	1	53	61	82	86
	1	16	30	44	68	81
	2	8	34	53	76	86
16	4	4	37	58	78	87
	8	2	37	55	77	85
	16	1	31	49	72	77

Table XI. Performance of the single-grid and multigrid method for test case 2 on the Meiko transputer system using a 128×128 CV grid and various processor configurations (EI mode)

Processor configuration			Single-grid method			Multigrid method		
n	n_x	n_y	Time (s)	E_n^{tot} (%)	No. of iterations	Time (s)	E_n^{tot} (%)	No. of iterations
1	1	1	29140[a]	100	1140	1131[a]	100	20
	1	4	7978	91	1160	335	84	22
4	2	2	7768	94	1144	307	92	21
	4	1	7985	91	1169	318	89	21
	1	8	4187	87	1178	196	72	23
	2	4	4018	91	1160	174	81	22
8	4	2	4033	90	1172	173	82	21
	8	1	4250	86	1193	224	63	28
	1	16	2256	81	1173	131	54	21
	2	8	2113	86	1177	103	69	22
16	4	4	2090	87	1186	100	71	23
	8	2	2147	85	1193	122	58	28
	16	1	2376	77	1282	145	49	28

[a] Estimated.

exchanged is lower for square subdomains than for stripes. For example, with a grid of $N \times N$ CVs, each stripe has two boundaries with $2N$ total points (except for the first and last stripes; their effect is negligible for large n). Using square subdomains results in four bondaries per subdomain with $4N/n^{1/2}$ points (except for boundary subdomains, where the multiplier four is replaced by

three or two). For computers with short set-up times, this gives shorter communication times. However, on computers with relatively long set-up times (like Suprenum), this arrangement may be less efficient, since the number of boundaries per subdomain (frequency of local communication) is increased from two to four.

The second effect has to do with global communication. In a ring configuration (stripes), messages are passed from one processor to its neighbour in one direction, so there are effectively $2(n-1)$ data transfers. For the array arrangement (square subdomains), global communication is parallel in one direction, and only for the last row it is sequential in the cross-direction, so that the total number of effective data transfers is $4(n^{1/2}-1)$. Since the amount of data transferred (e.g., when checking the convergence, it is usually only the residual sum i.e. only eight bytes), the set-up time is important. The fact that the global communication is proportional to n for the ring topology and to $n^{1/2}$ for the thorus is reflected in higher efficiencies for square subdomains.

The difference in the efficiencies for vertical and horizontal stripes of the same size is due to the alignment or non-alignment of stripes with the data storage arrangement (i.e. in one case, the data is contiguous, in the other case not). The unfavourable arrangement requires additional gather and scatter operations to prepare the data transfer.

The effects of grid size and number of processors on efficiency are in line with those presented for test case 1. The number of outer iterations in the multigrid algorithm does not remain the same when the number of processors is increased, as was the case in the first example. This could have been achieved by increasing the number of inner and outer iterations on the coarser grids; however, the efficiency would not be significantly affected.

6. PREDICTION OF PARALLEL EFFICIENCY

For a particular numerical algorithm, it is possible to create a communication model which allows prediction of parallel efficiency on different computers. This effort has been undertaken in the present case for the following reasons:

(1) to examine the influence of the hardware and system software parameters on the parallel efficiency of the present algorithm;
(2) to predict efficiencies for calculations with larger numbers of processors which are presently not available, in order to examine the suitability of the algorithm for massive parallelization;
(3) to check the influence of modifications of the algorithm on the efficiency.

The definition of parallel efficiency is, according to equation (12):

$$E_n^{\text{par}} = \frac{T_n^{\text{calc}}}{T_n^{\text{calc}} + T_n^{\text{com}}} = \frac{1}{1 + t_n^{\text{com}}/t_n^{\text{calc}}}. \tag{13}$$

To develop a model, it is necessary to express the calculation and communication times in terms of the hardware parameters.

The calculation time per outer iteration was defined earlier as $t_n^{\text{calc}} = N_n^{\text{cv}} \tau i_n$. The communication time per outer iteration, t_n, may be split into two parts:

$$t_n^{\text{com}} = t_n^{\text{loc}} + t_n^{\text{glob}},$$

where t_n^{loc} stands for the local and t_n^{glob} for the global communication times. The local communication involves only nearest neighbours. It can take place in parallel and is independent of the number of procesors (for a given subdomain grid and for n sufficiently large, to ignore the effect of

global boundaries). In the present algorithm, local communication is required at several stages, (see Figure 2):

(1) transfer of the coefficient A_P from the equations for U and V to enable the assembly of the pressure-correction equation;
(2) transfer of corrected velocities after the pressure-correction equation is solved;
(3) in the multigrid algorithm, transfer of restricted variable values;
(4) transfer of variable values after each inner (EI mode) or outer (EO mode) iteration.

For the EI mode of communication, t_n^{loc} can be expressed in terms of the most important parameters as follows:

$$t_n^{\text{loc}} = 2\left(\alpha + \sum_{i=1}^{N_{\text{var}}} N_i^{\text{it}}\right)\left(N_{\text{nb}} t^{\text{st}} + \sum_{l=1}^{N_{\text{nb}}} \frac{N_l^{\text{cv}} N_l^{\text{db}}}{R_{\text{tr}}}\right), \tag{14}$$

where N_{nb} denotes the number of neighbour processors (two for stripes and four for blocks in two-dimensional problems; two to six in three dimensions), N_{var} is the number of dependent variables, N_i^{it} is the number of inner iterations for variable i, t^{st} is the set-up time for data transfer initialization, N_l^{cv} is the number of CVs along interface l, N_l^{db} is the number of bytes per interface CV which need to be transferred and R_{tr} is the transfer rate in bytes per second. The factor α accounts for data transfers outside the inner iteration loop (as discussed above) and is equal to the number of variables whose values need be transferred (four to seven, depending on the configuration and algorithm).

The global communication consists of reporting global parameters (e.g. residual level) from each processor to the 'master', or broadcasting a message from the master to all other processors (e.g. whether to stop calculation or not). In general, one global two-way communication described above is performed after each inner iteration for each variable, and after each outer iteration. Another kind of global communication used in the present algorithm is broadcasting of the pressure-correction value at a location where pressure level is preset; see Figure 2. The global communication time can then be expressed (for two-dimensional problems) as:

$$t_n^{\text{glob}} = \left(\beta + 2\sum_{i=1}^{N_{\text{var}}} N_i^{\text{it}}\right)[(n_x - 1) + (n_y - 1)]\left(t^{\text{st}} + \frac{N_g^{\text{db}}}{R_{\text{tr}}}\right), \tag{15}$$

where n_x and n_y denote number of subdomains (processors) in x- and y-direction, respectively; β is the number of global communications at the outer iteration level and N_g^{db} is the number of bytes which need be transferred in one global communication. In three-dimensional problems, the square bracket would contain an additional term, $+(n_z - 1)$, where n_z is the number of subdomains in z-direction. It should be noted that an experienced user may reduce the global communication by specifying a fixed number of inner iterations (sweeps) for each variable, without checking convergence, thus eliminating the factor $\sum_{i=1}^{N_{\text{var}}} N_i^{\text{it}}$. The overall convergence need not be checked after each outer iteration. This increases the risk of doing more iterations than necessary, but may substantially cut communication time on systems with high set-up times.

The expressions for t_n^{loc} and t_n^{glob} can now be inserted into the expression for parallel efficiency, equation (13). By using the measured values of t^{st}, τ and R_{tr}, this equation can be used to predict the parallel efficiency of the given algorithm on a given computer as a function of the number of processors. In order to verify the model, the efficiencies were calculated and compared with the measured ones for test case 1 and various grid sizes; see Figure 7. The solid line drawn through the predictions is to guide the eye; the number of processors must be an integer! The kink at $n = 2$ is due to the fact that in this case each processor has only one neighbour and, thus, does only half the usual local communication.

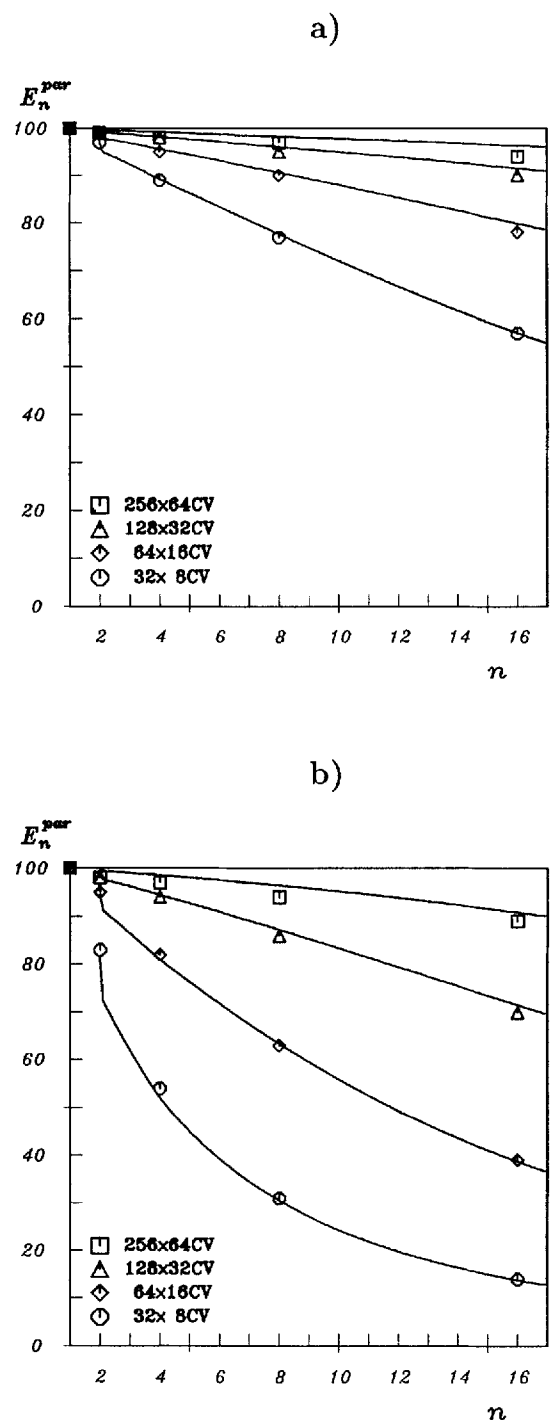

Figure 7. Comparison of measured and predicted parallel efficiency for the test case 1 and various grids, for the Meiko transputer system (a) and Suprenum (b)

As can be seen from Figure 7, the agreement between the predicted and measured efficiencies is very good (the discrepancy is only a few per cent). It is especially encouraging that the model works properly for two different computers with an order of magnitude difference in the communication parameter.

Having verified the model, the limiting factors for two-dimensional problems may be studied. First, the case of given grid size is analysed. The calculation time depends reciprocally on the number of processors used. The local communication time depends on the number of the CVs along subdomain boundaries. For stripes, it is independent of the number of processors while, for an array, it is inversely proportional to $n^{1/2}$. Therefore, the ratio $t_n^{\text{loc}}/t_n^{\text{calc}}$ is an (stripes) or $an^{1/2}$ (square blocks; not same a). The global communication time is proportional to either n (stripes, $n_x=n$ and $n_y=1$) or $n^{1/2}$ (square blocks, $n_x=n_y=n^{1/2}$), as discussed in the previous section. The ratio $t_n^{\text{glob}}/t_n^{\text{calc}}$ is then bn^2 or $bnn^{1/2}$. One can then write

$$E_n^{\text{par}}=\frac{1}{1+an+bn^2} \quad \text{or} \quad \frac{1}{1+an^{1/2}+bnn^{1/2}}, \tag{16}$$

where a and b are independent of n. The parallel efficiency thus approaches zero as $n\to\infty$ (but obviously no more processors than CVs can be used in the present algorithm).

For a constant number of processors and varying grid size (uniformly in both directions), the calculation time is proportional to the number of CVs. The local communication time is proportional to the square root of the number of CVs, and the global communication time remains constant. Thus, for a given shape of subdomains, the parallel efficiency can be expressed as

$$E_n^{\text{par}}=\frac{1}{1+a/(N_n^{\text{cv}})^{1/2}+b/N_n^{\text{cv}}}.$$

Note that the coefficients a and b are not the same as in equation (16) and do not depend on the grid size. As the number of CVs grows, the parallel efficiency for a given number of processors tends to one, as expected.

If the load per processor is kept constant ($N_n^{\text{cv}}=\text{const.}$) and the number of processors is increased as the grid size is increased—which is the most likely case in practice—then the calculation and local communication times remain constant. The global communication time increases with n so the parallel efficiency depends then on n as follows

$$E_n^{\text{par}}=\frac{1}{1+a+bn} \quad \text{or} \quad \frac{1}{1+a+bn^{1/2}},$$

for stripes and square blocks, respectively.

In three-dimensional problems, the grid may be partitioned in one, two or all three directions. In the first two cases, the limiting factors are the same as those given above for stripes and squares. In the case of subdivisions in three directions with an equal number of subdomains in each direction ($n_x=n_y=n_z=n^{1/3}$), the following limiting factors arise:

for given grid size

$$E_n^{\text{par}}=\frac{1}{1+an^{1/3}+bnn^{1/3}},$$

for given number of processors

$$E_n^{\text{par}}=\frac{1}{1+a/(N_n^{\text{cv}})^{1/3}+b/N_n^{\text{cv}}},$$

for given load per processor

$$E_n^{\text{par}} = \frac{1}{1 + a + bn^{1/3}}.$$

The limiting factors are, therefore, a function of the dimensionality of processor configuration and not of the grid dimensionality. The dependence on n is somewhat better for three- than for two-dimensional configurations (but obviously the former are applicable only to three-dimensional flow problems).

The above analysis shows that global communication is the limiting factor for a large number of processors. The efficiency can only be improved by modifying the global communication structure. For example, by implementing a binary tree topology (relatively easy for bus architectures), the global communication cost would be proportional to $\log n$. If global communication is not performed after every iteration, the factor b is reduced, but the overall trend remains the same.

Parallel efficiencies of the present algorithm were evaluated using the model described above for a grid of 1000×1000 CV, large numbers of processors and two computers (see Figure 8). The 60% level is reached on the Meiko transputer system with about 1000 processors and on Suprenum at about 450 processors (the maximum number of processors for Suprenum is actually 256). It is interesting that, for a small number of processors (up to 100) and this large grid, the Suprenum offers slightly higher efficiency than the Meiko transputer system, which seems contradictory to the results discussed before. This is due to its faster data transfer rate, see Table I, which becomes important when messages are large. With increasing number of processors, the message size reduces and the set-up time becomes dominant, causing the efficiency to deteriorate much faster than is the case with the Meiko computer.

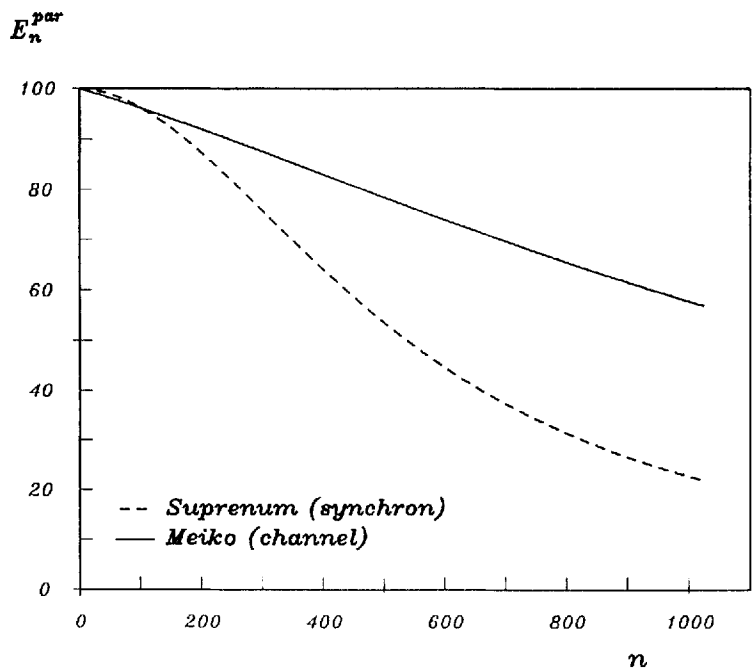

Figure 8. Predicted parallel efficiency for a grid of 1000×1000 CV and various numbers of processors, for the Meiko transputer system and Suprenum

Table XII. Comparison of computing times and number of iterations for the test case 2 and a 384×512 CV grid on various computers

Computer	No. of processors	No. of iterations	Comp. time (s)
Cray YMP	1	21	315
Meiko	48	22	382
Parsytec	48	22	527
	192	28	324

The unfavourable dependence of E_n^{par} on n could be avoided only by changing the computer architecture. For example, if the host (or any other processor) was provided access to the data in each processor's private memory, then the gathering or broadcasting could be performed while other processors are calculating. The negative effect of the global communication on the efficiency could also be eliminated if it took place simultaneously with calculation. Between two global communications there is normally a large number of computing operations, so the arrival of broadcasted information would be at most one iteration late. For a convergence check, this could be taken into account by adjusting the convergence criterion; for the pressure correction level at reference location, this would not matter, because at convergence it is supposed to be negligibly small everywhere.

Finally, calculations for test case 2 and a grid with 348×512 CV were performed on the Meiko transputer system with 48 processors, on the Parsytec transputer system with 48 and 192 processors and on the Cray YMP (no vectorization). Computing times and numbers of outer iterations are presented in Table XII. The estimated total efficiency of the parallel computing was about 80% with 48 processors and about 50% with 192 processors. The comparison of computing times is unfair; a vectorized code would run 5–10 times faster on Cray YMP. The main reason for using Cray YMP was to estimate the numerical efficiency, since it was the only computer available which could store the whole grid in single processor's memory.

In the analysis and test calculations presented above, the least efficient communication was used; the results should, therefore, be seen as a lower-limit case. Future parallel computers (and some present ones, which were not available to the authors) will offer one of the above mentioned possibilities for improving parallel efficiency, so that it is likely to get closer to 100%. The total efficiency will then depend mostly on numerical efficiency. Multigrid methods appear to retain their efficiency on the finest grid, at the expense of increased number of iterations on coarse grids, which indicates that implicit solution methods will run efficiently on massively parallel computers.

CONCLUSIONS

Results of test calculations with the parallelized version of an implicit multigrid finite volume code and theoretical analysis allow the following conclusions to be drawn:

(1) The efficiency of parallel computing of fluid flows using implicit finite volume algorithms and domain decomposition technique is influenced by three major parameters:
   (a) Increase in the number of inner and outer iterations needed to converge due to the decoupling of subdomains; this is characterized by the *numerical efficiency*;
   (b) communication time needed to transfer boundary data between processors (local

communication) and for monitoring of convergence (global communication); this is characterized by the *parallel efficiency*;

(c) idle times of processors due to uneven load; this is characterized by the *load-balancing efficiency*.

The total efficiency is equal to the product of these three parameters, and provides an important criterion for evaluating parallel algorithms and computers. The parallel efficiency alone is not sufficient for this judgement.

(2) The analysis and numerical experiments show that parallel performance becomes less efficient as the number of processors grows at constant load, due to the global communication. The critical hardware parameter is the ratio of communication speed to the calculation speed, which is dominated by the set-up time for initiating communication between processors.

(3) The test calculations show that the numerical efficiency of the multigrid algorithm is not affected more by the domain decomposition than is the efficiency of the single-grid algorithm. Both are high when boundary data are exchanged between processors after each inner iteration. The parallel efficiency, on the other hand, is worse for the multigrid algorithm due to the use of coarse grids with a large number of processors and more communication per outer iteration. In spite of that, the total computing times with the parallel multigrid algorithm are orders of magnitude shorter than with the single-grid algorithm.

(4) A theoretical model of the parallel efficiency was developed and shown to predict the measured efficiencies well. It takes into account the number of processors used, the communication patterns in the algorithm and the hardware characteristics including computing time per floating point operation, set-up time for communication and the rate of data transfer between processors.

The results of this study show that implicit solution methods can be adapted for parallel processors using domain decomposition techniques, provided that the communication is not too slow compared to the calculation. It is expected that these findings remain valid for three-dimensional applications, and that the efficiencies will be higher for complex geometries and flow phenomena, since the number of floating-point operations per control volume and iteration will be much higher than in the cases studied here, while the amount of communication will remain more or less the same.

ACKNOWLEDGEMENTS

The Commission of the European Communities provided via 'Parallel Computing Action' a part of the Meiko Computing Surface used in this study; the Deutsche Forschungsgemeinschaft provided the Parsytec Supercluster within its program 'Strömungssimulation mit Hochleistungsrechnern' and financial support through the 'Sonderforschungsbereich 182'; the Computer Science Department of the University of Erlangen (IMMD4 and IMMD3) provided access to the Suprenum and Meiko computers; A. Bohm, J. H. Ferziger and G. Horton helped with many discussions. The authors thank all of them for their support.

REFERENCES

1. H. L. Stone, 'Iterative solution of implicit approximations of multi-dimensional partial differential equations', *SIAM J. Numer. Anal.*, **5**, 530 558 (1968).
2. S. V. Patankar and D. B. Spalding, 'A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows', *Int. J. Heat Mass Transfer*, **15**, 1787–1806 (1972).

3. M. Perić, R. Kessler and G. Scheuerer, 'Comparison of finite-volume numerical methods with staggered and colocated grids', *Comput. Fluids*, **16,** 389–403 (1988).
4. M. Hortmann, M. Perić and G. Scheuerer, 'Finite volume multigrid prediction of laminar natural convection: bench-mark solutions', *Int. j. numer. methods fluids*, **11,** 189–207 (1990).
5. P. Bastian and G. Horton, 'Parallelization of robust multi-grid methods: ILU factorization and frequency decomposition method', in W. Hackbusch and R. Rannacher (eds), *Notes on Numerical Fluid Mechanics, Vol. 30*, Vieweg, Braunschweig, 1989, pp. 24–36.
6. G. Fox *et al., Solving Problems on Concurrent Processors, Vol. 1*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
7. R. Hempel and A. Schüler, 'Experiments with parallel multigrid algorithms using the SUPRENUM Communications Subroutine Library', *GMD-Studien*, No. 141, 1988.