Parallel algorithms for DNS of compressible flow

Martin Streng, Hans Kuerten, Jan Broeze and Bernard Geurts
Department of Applied Mathematics, University of Twente
P.O.Box 217, 7500 AE Enschede, The Netherlands

Abstract

We indicate that the use of higher order accurate spatial discretization is necessary to obtain sufficiently accurate DNS for the validation of subgrid models in LES. Furthermore, we pay attention to the efficiency of the implementation of these discretizations on several parallel platforms. In order to illustrate this, we consider compressible flow over a flat plate. We give *a priori* test results for LES of this flow.

1 Introduction

One of the most challenging problems in Computational Fluid Dynamics (CFD) is the accurate and efficient simulation of turbulent flows for relevant industrial applications. The behaviour of these flows is governed by the Navier-Stokes equations. However, because these applications usually involve complex geometries and flow-fields, the computational resources required for directly solving the Navier-Stokes equations are far beyond the resources which will be available in the foreseable future. In this paper we will focus on turbulent compressible flow-problems in simple geometries. In order to tackle these problems with presently available computers, three different aspects must be considered: the modelling of turbulent flows, the numerical methods used to perform calculations with these models, and the implementation of these methods on suitable computer platforms.

As remarked above, direct solution of the Navier-Stokes equations (DNS) is impossible for relevant industrial applications, due to the high computational requirements. Therefore, one might use instead the Reynolds averaged Navier-Stokes (RaNS) equations in which only the statistically stationary flow is calculated and the effects of turbulence are modelled by a so-called turbulence model. However, this leads in general to quite inaccurate results since the presently available turbulence models are inadequate for more complicated flow phenomena like shock-boundary layer interaction and massive separation. A solution to this problem could be provided by Large Eddy Simulation (LES). In LES only the large eddies are calculated, while the effects of the smaller eddies, which are thought to be universal and not geometry-dependent, are described by a subgrid model.

However, before LES can be used as a tool in flow simulation, the subgrid model has to be systematically validated. This validation is usually carried out by comparing LES results with filtered DNS results for simple geometries and fairly low Reynolds numbers. In Section 2 we present *a priori* test results for LES of compressible flow over a flat plate for various subgrid models, including eddy-viscosity models, the similarity model and dynamic models. In the future also *a posteriori* tests will be carried out fot this flow, as has been done e.g. by Vreman et al. [1] for the compressible mixing layer.

The numerical methods to perform the DNS are discussed in Section 3. The a priori test results are based on DNS performed using a second-order finite volume spatial discretization. It is indicated that the use of higher order spatial discretizations makes it possible to obtain more accurate DNS results. However, the use of higher order central differencing discretizations, without numerical dissipation, is not without trouble. Besides the occurrence of stability problems, higher order discretizations lead to wide stencils, which, in combination with a domain-decomposition strategy, seriously affects the parallel efficiency of the resulting algorithm.

In Section 4 the parallel efficiency will be illustrated using some implementations of the DNS solver on various parallel platforms, including distributed as well as shared memory systems, and a mixture of these types. Since many parallel platforms use cache-based processors, we consider some aspects of implementation of the flow-solver on these processors. We show that careful use of cache in the implementation of our type of discretizations can lead to considerable performance gain.

2 Modelling of turbulent flow

The equations describing compressible flow are the well known Navier-Stokes equations, which represent conservation of mass, momentum and energy:

$$\begin{aligned} \partial_t\rho &+ \partial_j(\rho u_j) = 0 \\ \partial_t(\rho u_i) &+ \partial_j(\rho u_i u_j) + \partial_i p - \partial_j \tau_{ij} = 0 \\ \partial_t e &+ \partial_j((e+p)u_j) - \partial_j(\tau_{ij}u_i - q_j) = 0 \end{aligned} \tag{1}$$

Here the symbols ∂_t and ∂_j are abbreviations of the partial differential operators $\partial/\partial t$ and $\partial/\partial x_j$ respectively. The components of the velocity vector are denoted by u_i, while ρ is the density and p the pressure which is related to the total energy density e by:

$$p = (\gamma - 1)\{e - \frac{1}{2}\rho u_i u_i\} \tag{2}$$

in which γ denotes the adiabatic gas constant. The viscous stress tensor τ_{ij} is a function of temperature T and velocity vector $\mathbf{u}$

$$\tau_{ij}(T, \mathbf{u}) = \frac{\mu(T)}{Re}(\partial_j u_i + \partial_i u_j - \frac{2}{3}\delta_{ij}\partial_k u_k) \tag{3}$$

where $\mu(T)$ is the dynamic viscosity for which we either use Sutherland's law for air or treat it as a constant. In addition q_j represents the viscous heat flux vector, given by

$$q_j(T) = -\frac{\mu(T)}{(\gamma-1)RePrM^2}\partial_j T \tag{4}$$

where Pr is the Prandtl number. Finally, the temperature T is related to the density and the pressure by the ideal gas law

$$T = \gamma M^2 \frac{p}{\rho} \tag{5}$$

These governing equations have been made dimensionless by introducing a reference length L_0, velocity u_0, density ρ_0, temperature T_0 and viscosity μ_0. The values of the Reynolds number $Re = (\rho_0 u_0 L_0)/\mu_0$ and the Mach number $M = u_0/a_0$, where a_0 is a reference value for the speed of sound, are given separately.

A Direct Numerical Simulation (DNS) is based on a discretisation of (1) whereas the governing equations for large eddy simulation (LES) are obtained by applying a spatial filter to these equations. A filter operation extracts the large scale part $\bar{f}$ from a quantity f:

$$\bar{f}(\mathbf{x}) = \int_\Omega G_\Delta(\mathbf{x}, \xi) f(\xi) d\xi \tag{6}$$

where Ω is the flow domain and Δ denotes the filter width of the kernel G which is assumed to be normalized, i.e. the integral of G over Ω equals 1 independent of $\mathbf{x}$. For compressible flow Favre [2] introduced a related filter operation $\tilde{f} = \overline{\rho f}/\bar{\rho}$.

The filtered Navier-Stokes equations contain so-called subgrid-terms, which cannot be expressed in the filtered flow variables, and have to be modelled with subgrid-models. In this paper we will mainly focus on the modelling of the subgrid-terms in the momentum equations, which can be expressed in the turbulent stress tensor, defined as

$$\bar{\rho}\tau_{ij} = \overline{\rho u_i u_j} - \overline{\rho u_i}\,\overline{\rho u_j}/\bar{\rho} = \bar{\rho}(\widetilde{u_i u_j} - \tilde{u}_i\tilde{u}_j), \tag{7}$$

where $\tilde{\mathbf{u}}$ is the filtered velocity vector. This turbulent stress tensor has several algebraic properties which can be used in the construction and qualification of subgrid-models [3, 4]. Expressions for the subgrid-terms in the energy equation can be found in ref. [5]. They can be neglected in simulations at low Mach numbers, but have to be modelled at high Mach numbers.

In total six models for the turbulent stress tensor τ_{ij} as it appears in the subgrid-terms in the momentum equations will be investigated and compared in this paper. The first subgrid-model is the Smagorinsky model

$$\bar{\rho}\tau_{ij}^{(1)} = -\bar{\rho}C_S^2\Delta^2|\tilde{S}|\tilde{S}_{ij}, \tag{8}$$

where $\tilde{S}^2 = \frac{1}{2}\tilde{S}_{ij}^2$ with $\tilde{S}_{ij}$ the compressible strain rate, based on the Favre-filtered velocity. C_S is the Smagorinsky constant, which we choose equal to 0.17 as suggested in literature. Δ denotes the filter width, which separates the resolved and subgrid-scales. The major short-coming of the Smagorinsky model is its excessive dissipation in regions where the flow is laminar [6]. The similarity model, formulated by Bardina *et al.* [7], is based on a similarity assumption. Application of the definition of $\bar{\rho}\tau_{ij}$ to the filtered variables $\bar{\rho}$ and $\overline{\rho u_i}$ yields the similarity model [7]:

$$\bar{\bar{\rho}}\tau_{ij}^{(2)} = \overline{\bar{\rho}\tilde{u}_i\tilde{u}_j} - \overline{\bar{\rho}\tilde{u}_i}\,\overline{\bar{\rho}\tilde{u}_j}/\bar{\bar{\rho}}. \tag{9}$$

The gradient model is derived with use of Taylor expansions of the filtered velocity [8]. The lowest order term in Δ in this expansion can be proposed as subgrid-model:

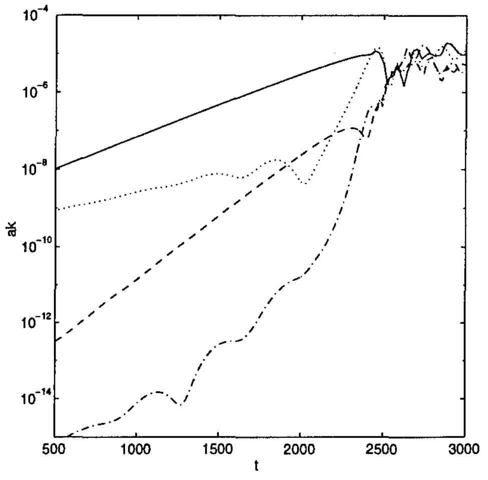
$$\bar{\rho}\tau_{ij}^{(3)} = \frac{1}{12}\bar{\rho}\Delta^2(\nabla\tilde{u}_i)(\nabla\tilde{u}_j). \qquad (10)$$

The similarity and gradient model correlate much better with the turbulent stress tensor than the Smagorinsky model (see [9] and section 2.1). However, while the Smagorinsky model is too dissipative in transitional regions, the similarity and gradient model are not sufficiently dissipative in turbulent regions.

The dynamic procedure overcomes the excessive dissipation of the Smagorinsky model and adds sufficient dissipation to the similarity and gradient models. We consider three dynamic models. The dynamic eddy-viscosity model [3] is obtained when the model constant C_S in the Smagorinsky model is replaced by a coefficient which is dynamically obtained and depends on the local structure of the flow. In order to calculate the dynamic coefficient $\tau_{ij}^{(1)}$ is substituted in the Germano identity, which is a relation between the turbulent stress tensor for different filter widths [3]. The second dynamic model is the dynamic mixed model, in which a relatively accurate representation of the turbulent stress by the similarity model and a proper dissipation provided by the dynamic eddy-viscosity concept are combined [10]. The dynamic model coefficient is obtained by substitution of the base mixed model, $\tau_{ij}^{(2)} + \tau_{ij}^{(1)}$, in the Germano identity. Another dynamic model is the dynamic Clark model [11]. In this case the base model is the Clark model, $\tau_{ij}^{(3)} + \tau_{ij}^{(1)}$, and the model coefficient C_S is obtained by substitution of this model in the Germano identity.

2.1 Results

We consider flat plat flow at $Re = 1000$ based on the initial displacement thickness δ_* and the other reference scales are equal to the initial far-field values. We choose $M = 0.5$ and consider a temporal simulation in a cubic domain of size 30. A forcing term corresponding to the compressible similarity solution of the boundary layer equations is added. The mean initial field also equals this similarity solution, to which the dominant 2D mode and a pair of equal and oblique 3D modes are added with amplitude 10^{-3} and amplitude-ratios $(1/2, 1/4, 1/4)$ respectively. For validation purposes the linear growth rates of the instabilities were recovered with a relative error well within 1 percent on a grid with 128^3 cells, uniform in the stream- and spanwise directions and clustered near the isothermal, no-slip wall in the normal direction. A second order accurate finite volume method was used.



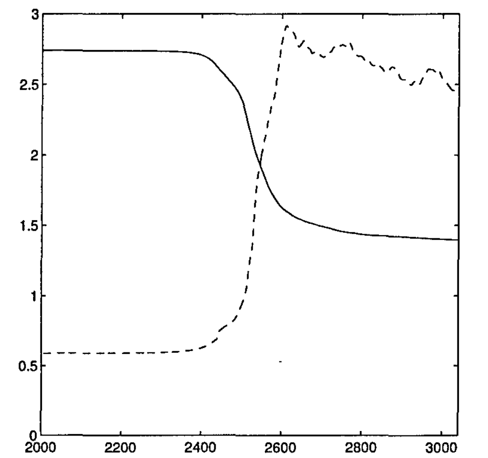


Figure 1: Modes of kinetic energy (a) [(1,0):solid, (2,0): dashed, (1,1): dotted, (2,2): dash-dotted] and shape-factor (solid), skin-friction (dashed) versus time t (b)

Results from a DNS on 128^3 cells are shown in Figure 1. The persisting symmetry in the spanwise direction was exploited in order to reduce the computational effort. The evolution of the amplitude of some modes of the kinetic energy (Fig. 1) clearly displays the initial linear regime with an exponential growth of the instabilities. The corresponding large-scale structures which emerge subsequently interact in the nonlinear regime and give rise to a rapid transition in which many modes become simultaneously important. A broad spectrum is generated and a developed turbulent flow results in which the individual modes display an erratic time-dependence. To represent this scenario in a different way, the shape-factor and the skin-friction are shown in Figure 1. The resolution is adequate in the linear and transitional stages with a fall off of 10 decades or more in the spectrum of the kinetic energy. However, at the onset of turbulent flow and in the developed stages a fall off of no more than 6-7 decades was observed. Hence, the results in the turbulent regime are expected to be only qualitatively correct and further grid refinement is needed.

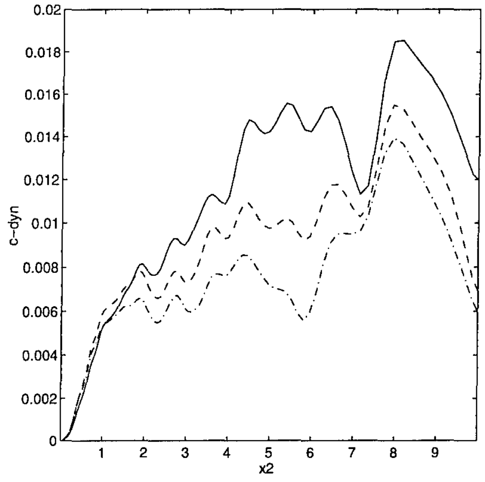


Figure 2: Dynamic coefficients : Germano (solid), dynamic mixed (dashed) and dynamic Clark (dash-dotted).

In order to obtain a first impression of the quality of the various subgrid models for this flow we focus on the correlation between $\bar{\rho}\tau_{12}$ and the corresponding modelled component of the turbulent stress tensor. We use a filter-width equal to four grid-cells and a special filtering near the wall which prevents the filter to extend inside the wall. The models are tested both in the transitional and in the turbulent regime. The similarity- and gradient model as well as the dynamic mixed and dynamic Clark model show a high correlation of about 0.9. The Smagorinsky and dynamic eddy-viscosity models show a poor correlation of about 0.3. The eddy-viscosity contribution in the dynamic mixed and dynamic Clark model does not destroy the high correlation. In Figure 2 we compare the dynamic coefficients for the three dynamic models at $t = 2700$. The coefficients are averaged over the homogeneous directions. We observe that the Germano coefficient is larger than the coefficient associated with the other two dynamic models. Moreover, all coefficients drop to zero in the near-wall region which is appropriate for wall-bounded shear layers.

3 Numerical method

As has been remarked in the previous section, the DNS results in the turbulent regime are expected to be only qualitatively correct, and further grid refinement is needed. However, the number of grid-cells used is already fairly large for presently available computer resources. Instead of refinement, we presently consider the use of higher order discretization methods. The aim is to obtain a more accurate DNS with a moderate number of points. However, this is not without problems. One drawback is that high order methods lead to wide stencils, which decreases the parallel efficiency of the resulting code, as we will see in the next section. Another problem associated with these methods is that the discretisation of the convective and the viscous flux must be carefully constructed in order to avoid instabilities. This is especially present in central differencing methods, and is not only related to the occurrence of π-modes, but also to adequate damping of aliasing errors.

3.1 Spatial discretization

Consider an orthogonal grid with points $x_{i,j,k}$, which is uniform in x and z direction. We use the following central differencing discretization of the $\frac{\partial}{\partial x}$-operator:

$$(\frac{\partial f}{\partial x})_{ijk} = \sum_{n=-d}^{d} w_{j,n}^{diff} a_{i+n,j,k}, \qquad (11)$$

where

$$a_{ijk} = \sum_{n,m=-d}^{d} w_{j,n,m}^{av} f_{i,j+n,k+m}. \qquad (12)$$

Here the weights w^{diff} are derivative weights, and w^{av} are average weights. Due to the uniformity in x and z direction they only depend on j. The quantities a represent the average of the function f over a stencil in $j-k$ direction. For the convective flux we use a stencil with N_c points, and the weights w^{av} are constructed such that π modes in the j and k direction are filtered out, and moreover that polynomials up to degree $N_c - 1$ are invariant under the averaging. The derivative weights w^{diff} are such that polynomials up to degree N_c are exactly differentiated. The resulting discretization has order N_c on uniform grids. The π-modes in i-direction are damped by the viscous derivative. The viscous flux is discretized using repeated differentiation. The inner derivative is calculated on a staggered grid. Both the inner and the outer derivatives are discretized analogously as in the convective flux, on N_v points, except that now π-modes are not filtered out. Both derivatives are then of order $N_v - 1$, but due to symmetry, on a uniform grid, the viscous flux is discretized up to order N_v.

Due to the nonlinearity in the convective flux, high frequency modes arise from a low-frequency initial state. In physical reality, these are damped by the viscous effects in the fluid. In the numerical simulation, however, two difficulties arise. The first is that both the convective and the viscous flux are calculated inaccurately. In our central differencing discretisations, on relatively coarse grids, a situation may arise in which the numerical viscous terms do not have enough dissipation to damp the numerical convective terms, giving rise to instabilities. The second difficulty is that due to the finite grid-spacing, there is a maximum wavenumber which can be represented on the grid. Modes with a higher wavenumber appear as low-frequency modes on the grid. Therefore, numerically, the effective energy contained in the low-frequency modes can be increased during the onset of turbulence. One remedy could be to take a grid that is sufficiently fine to represent the highest mode which due to physics would emerge in the simulation. Another possibility is to use upwind-biased discretizations of the convective flux, as has been done by Rai and Moin [12]. We have used a discretisation of the viscous flux with a wider stencil than necessary to achieve the desired order of accuracy. In this way we constructed a better approximation of the viscous flux. As an example, we were able to calculate a full transition to turbulence on 96^3 points using a fourth order method on a 5^3-points stencil for the convective flux, and repeated application of a fourth order method on 6^3 points for the viscous flux, resulting in an 11^3-points stencil, whereas repeated application of a 4^3 points operator for the viscous flux on this grid failed. At this moment, further investigation is needed to understand this phenomenon more clearly.

The DNS mentioned in the previous section has been calculated at Mach number 0.5. In the future we intend to perform DNS at higher Mach numbers. For that purpose we need to be able to capture shocks. This can be done by switching to upwind discretizations in the presence of a shock, which has been applied succesfully to the supersonic compressible mixing layer, cf. ref. [13]. In that application a fourth order central difference operator has been used for the convective term, which was replaced by a third order accurate upwind scheme in the presence of a shock. See Figure 3. In this way it is possible to capture time-dependent shocks which appear spontaneously after the transition to turbulence.

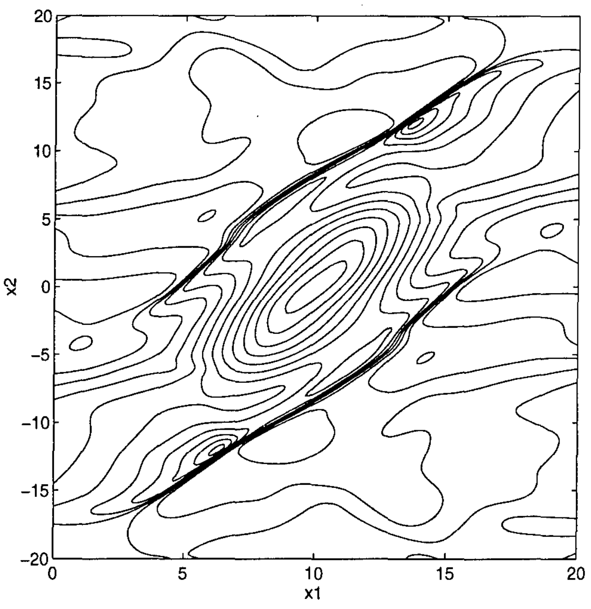


Figure 3: Shock-capturing in 3D turbulent mixing-layer.

3.2 Time integration

For the time integration of the resulting discretized equations we use an explicit 4-stage Runge Kutta method. We also studied the use of a second-order accurate implicit method. The system of equations resulting from the implicit discretization is solved by means of pseudo-time stepping and accelerated by local pseudo-time stepping and a nonlinear multigrid technique. Since we use central spatial discretizations and no artificial dissipation is added to the equations, the smoothing method is less effective than in the traditional use of multigrid in steady-state calculations. In the laminar regime and in the first stages of turbulence the implicit method provides a speed-up of a factor of 2 relative to the explicit method on a relatively coarse grid (64^3). At increased resolution this speed-up is enhanced correspondingly. See [14].

4 Parallel implementation of the explicit solver

In this section we consider some implementational aspects of the explicit solver. We use a simple domain-decomposition technique to obtain an implementation on a parallel computer. This is explained in the first subsection. In the next subsection we discuss how the parallel efficiency of the resulting code depends on the spatial discretization. We distinguish between the *intrinsic* efficiency of an algorithm, and the *hardware* efficiency. The former is related to the algorithm only, whereas the latter tells us how good a certain algorithm performs on certain hardware. The quantity which is usually called the efficiency is the product of these efficiencies. We show that the intrinsic efficiency of the algorithm decreases as the order of the spatial discretization increases. We illustrate these concepts by some performance results obtained from implementations on 3 different parallel machines, viz. the Cray T3d, the Intel Paragon and the SGI Power Challenge array. Closely related to the concept of efficiency is the scalability. We discuss the scalability in the sense of Amdahl and Gustafsson (see e.g. ref. [15]).

4.1 Domain decomposition

Suppose our computational domain consists of $N_x \times N_y \times N_z$ gridpoints. This domain is divided into $B_x \times B_y \times B_z$ blocks. For a distributed memory computer, we assume that each block is allocated on a separate processor. If the total size of the stencil used for the discretisation is $(2d+1)^3$ (recall that we use central differences, cf. (11),(12)), then a point which has a distance less than $d+1$ grid-points from the boundary of a block not coinciding with the boundary of the physical domain, is called an interior boundary point. This definition can easily be extended to other discretisation methods. For the computation of the fluxes for the interior boundary points, some values of the flow-quantities which reside on processors dealing with neighbouring blocks are needed. To store these quantities, each block is dressed with d dummy-layers. In order to retain the second-order accuracy of the time-integration method, at each stage in the Runge-Kutta time-integration, these dummy-layers have to be transferred between the various processors. It may be clear that the amount of communication increases with the size of the stencil.

Not only the amount of communication is affected by the size of the stencil, but also the number of floating point operations increases with increasing stencil-size. To see why, recall the general form of the $\frac{\partial}{\partial x}$-operator, eq. (11)–(12). This derivative is computed as a one-dimensional derivative acting on two-dimensional averages over y and z. For the derivative in an internal boundary point these averages have to be computed for points in the dummy-layers as well. But these averages are also computed by the processors dealing with the neighbouring block in order to contribute to the $\frac{\partial}{\partial x}$ derivative of some points in that block. For a discretization on a stencil with $N_x \times N_y \times N_z$ points, careful counting reveals that the number of floating-point operations for the computation of one derivative is

$$(3N_xN_yN_z + 4dN_yN_z + 2dN_xN_z + 4d^2N_z)(2d-1).$$

Note that this expression is not symmetric in N_x, N_y, N_z. For the other derivatives the discrete averaging and differentiation operators can be applied in such an order that the same expression is valid. In the case $N_x = N_y = N_z = N$, this reduces to

$$(3N^3 + 6N^2d + 4d^2N)(2d-1). \qquad (13)$$

Now consider e.g. a given partition of the computational domain into B^3 equal blocks, each containing $(N/B)^3$ points. Then the total number of floating-point operations to compute a $\frac{\partial}{\partial x}$ for all grid-points is

$$3((\frac{N}{B})^3 + 6(\frac{N}{B})^2d + 4d^2\frac{N}{B})(2d-1)B^3,$$

which is obviously greater than (13).

4.2 Parallel efficiency

To quantify the considerations of the previous paragraph, we define the concept of intrinsic efficiency. Consider a given partition of the computational domain into $B_x \times B_y \times B_z$ blocks. Denote the total number of floating point operations for a given number of timesteps by $f(B_x, B_y, B_z)$. Then the intrinsic efficiency σ_{intr} is given by

$$\sigma_{\text{intr}} = \frac{f(1,1,1)}{f(B_x, B_y, B_z)}. \qquad (14)$$

Note that, on a shared memory machine, if we use fine-grained parallellism (on do-loop level), we could define $\sigma_{\text{intr}} = 1$.

We can estimate the dependence of the intrinsic efficiency on the size of the stencil just by counting the number of floating-point operations for various block-sizes (by using expressions like (13)). In Figure 4 this has been done for several central differencing discretizations, using equal shapes and sizes for all blocks. From the pictures it can be seen that the efficiency decreases rapidly if the stencil-size grows. Due to the wider stencil, application of higher-order discretizations results in more floating-point operations, but this performance penalty is even more severe on distributed memory systems, where also a decrease of parallel performance occurs. As an example, consider a central differencing second order $\frac{\partial}{\partial x}$ operator on a 3-point stencil as compared to a central differencing fourth order $\frac{\partial}{\partial x}$ operator on a 5-point stencil. To compute the former derivative on a single-cpu machine costs approximately $5/9 \approx 0.56$ times of the time to compute the latter, whereas on e.g. a $64 \times 64 \times 32$ grid and 128 processors on a distributed memory machine this ratio is approximately 0.33.

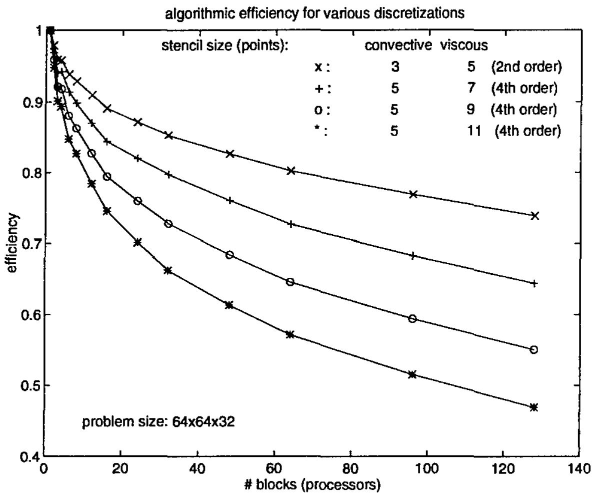


Figure 4: Intrinsic efficiency for various spatial discretizations

The intrinsic efficiency deals with the parallelizability of a given algorithm, regardless of any machine. In fact it gives the maximum speed-up that can be achieved for the algorithm. In a real implementation the speed-up will be less, due to e.g. the finite bandwidth of the machine. To quantify this, we now define the hardware efficiency σ_{hw}. Suppose the CPU time to perform a certain number of timesteps on one processor using one block is $T(1,1,1)$. Then, using $B_x B_y B_z$ processors, the CPU time cannot be shorter than

$$\frac{T(1,1,1)}{B_x B_y B_z \sigma_{\mathrm{intr}}(B_x, B_y, B_z)}.$$

In general, due to the finite communications bandwidth of the machine, the simulation will last longer, say $T(B_x, B_y, B_z)$ seconds. Then the hardware-efficiency σ_{hw} is

$$\sigma_{\mathrm{hw}} = \frac{T(1,1,1)}{T(B_x, B_y, B_z) B_x B_y B_z \sigma_{\mathrm{intr}}(B_x, B_y, B_z)}. \tag{15}$$

The traditional (total) efficiency σ is the product

$$\sigma = \sigma_{\mathrm{hw}} \sigma_{\mathrm{intr}}. \tag{16}$$

Note that, in general, these efficiencies not only depend on the number of blocks in each direction, but also on the number of points per block in each direction, i.e. on the actual shape of the blocks. This is not only due to the ratio of interior boundary points as compared to the interior points of each block, but also because many processors perform better on long inner loops in the code, due to vectorisation or pipelining.

The efficiency σ is related to scalability in the sense of Amdahl, meaning that a problem which is solved on one processor in T_1 seconds is solved on P processors in $T_1/P\sigma$ seconds. We define one notion of efficiency related to scalability in the sense of Gustafson. Suppose we solve a problem with N gridpoints on one processor in T_1 seconds, and a problem with PN gridpoints in T_P seconds. Then the efficiency σ_G is

$$\sigma_G = \frac{T_P}{T_1}. \tag{17}$$

These concepts are illustrated in Figure 5. Here we performed 5 timesteps on a $64 \times 64 \times 32$ grid, with a 5 point central differencing discretization of the convective flux, and a repeated application of a four-point central differencing for the viscous flux, resulting in a total stencil containing $7 \times 7 \times 7$ points. Plotted are the intrinsic efficiency and the total efficiency. Because it was not possible to execute the program on 1 or 2 CPUs on the Paragon, the efficiencies are based on the timings for the 4-processor run. We used 2 different distributed memory machines, viz. the Cray T3d and the Intel Paragon. On these machines, explicit message-passing has been employed. The actual CPU-times for the runs are tabulated in Table 1. A dash indicates that it had not been possible to perform the run on the indicated number of processors, either because the processors do not have enough memory (in the case of 1 and 2 processors on the Paragon) or because the indicated number of processors was not available on that machine. The CPU times are depen-

# proc.	T3d	Paragon
1	207.5	–
2	109.6	–
4	58.3	90.6
6	42.7	65.2
8	33.4	50.3
12	23.8	36.8
16	17.1	27.4
24	13.5	20.4
32	9.9	15.7
48	7.5	11.7
64	6.0	11.1
96	4.8	7.6
128	3.8	–

Table 1: CPU times in seconds (averaged over several block-divisions).

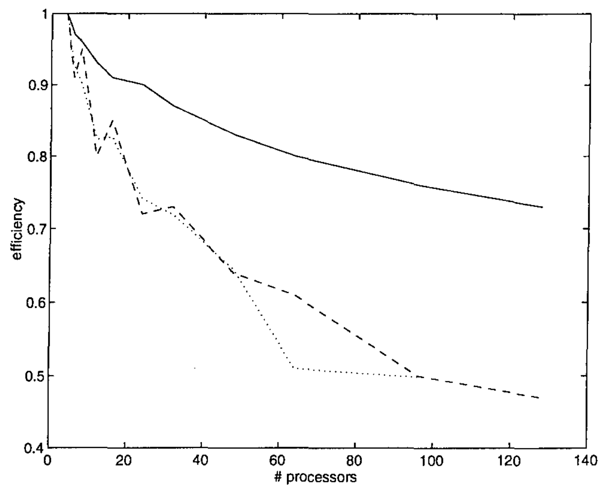


Figure 5: Efficiency for the T3d (dashed) and the Paragon (dotted). The solid line is the intrinsic efficiency.

dent on the actual shape of the blocks. Therefore in Table 1 we averaged over some block-divisions which give roughly the same (approximately best) CPU-time. This dependency is illustrated in Table 2 for the case of 8 blocks. All timings are accurate to about 5 %. It can be seen that subdivisions with an equal number of blocks in all directions are optimal. In general, better subdivisions are obtained by using fewer blocks in x-direction. This is partly due to the algorithm, since an asymmetry is introduced by the sequence of averaging-operators in the derivative-calculations, and partly due to software-pipelining in the processors, which is reflected in the megaflop-rates (between parentheses).

$B_x \times B_y \times B_z$	T3d	Paragon	
$1 \times 1 \times 8$	34.9 (77)	58.0 (47)	335
$1 \times 8 \times 1$	34.3 (82)	54.4 (52)	353
$8 \times 1 \times 8$	39.2 (77)	67.3 (45)	378
$1 \times 2 \times 4$	32.4 (80)	50.9 (52)	323
$1 \times 4 \times 2$	31.6 (83)	50.0 (53)	328
$2 \times 1 \times 4$	33.0 (79)	52.9 (50)	327
$2 \times 4 \times 1$	31.7 (84)	51.5 (53)	335
$4 \times 2 \times 1$	32.9 (80)	54.6 (50)	327
$4 \times 1 \times 2$	32.9 (82)	55.4 (49)	339
$2 \times 2 \times 2$	31.1 (84)	50.3 (53)	325

Table 2: CPU times for various subdivisions into 8 blocks. Between parentheses the Mflop-rates. The last column is the number of millions of floating point-operations to be performed for each block.

From the pictures it can be seen that on the T3d and the Paragon, the machine efficiency is somewhat lower than the algorithmic efficiency. This means that increasing the algorithmic efficiency by e.g. exchanging information between the processors after every calculation of averages will not result in a substantially faster execution of the code. Further, all efficiencies eventually approach zero as the number of processors approaches infinity. It can be shown (using expressions like (13)) that the intrinsic efficiency drops as $B^{-2/3}$, where B is the total number of blocks. However, σ_G remains nearly constant, as is shown in Table 3. Here each block contains $32 \times 16 \times 16$ points. From this table it follows that, using this algorithm, doubling the size of the problem and the number of processors results in equal computation times. This can also be shown if in (17) the times T_P and T_1 are calculated as ideal, i.e. assuming no communications delays. Then $\sigma_G = 1$.

$B_x \times B_y \times B_z$	T3d	Paragon
$1 \times 2 \times 1$	16.2 (21.1)	26.9 (12.7)
$1 \times 4 \times 1$	16.3 (42.1)	27.0 (25.4)
$1 \times 4 \times 2$	16.4 (83.6)	27.3 (50.2)
$2 \times 4 \times 2$	16.5 (166)	27.6 (99.4)
$2 \times 8 \times 2$	16.5 (332)	27.4 (200)
$2 \times 8 \times 4$	16.6 (661)	27.7 (396)
$2 \times 8 \times 6$	16.6 (991)	27.8 (592)
$4 \times 8 \times 4$	16.6 (1322)	–

Table 3: CPU times and Megaflop-rates (between parentheses) for increasing domain-sizes illustrating that σ_G remains approximately constant.

From the above results it can be concluded that the T3d and the Paragon show comparable efficiencies for this algorithm, the T3d being about 40 % faster.

Besides the implementation on the T3d and the Paragon, we have made a preliminary implementation on the SGI Power Challenge Array. This machine consists of 4 nodes each comprised of a 16-CPU shared memory parallel machine. We used explicit message-passing between the nodes. On each node, fine-grained parallelism has been employed using the vendor-supplied parallelizing compiler. The combination of fine-grained parallelism and explicit message passing is not entirely trivial. On the one hand, using fine-grained parallelism results in an algorithmic efficiency of 1, since no additional floating-point operations are introduced. Therefore, this form of parallelism seems to be promising at first sight. On the other hand, however, parallelizing a do-loop containing only a few iterations (in the order of magnitude of the number of grid-points in one directions) causes much system-overhead, and seriously affects pipelining efficiency. Moreover, suboptimal speedup can arise due to the cache-coherency mechanism. The use of explicit message-passing has two disadvantages, namely an algorithmic efficiency less than one, and usually a slow data-transfer. The advantage of explicit message-passing as compared to fine-grained parallelism is that parallelization takes place on a (much) higher level, leading to less system overhead.

As an example, consider a problem with $64 \times 64 \times 32$ grid-points (the same as discussed above). With 4 processors on one node working on one block, this yields an execution time of 23 seconds for 5 Runge-Kutta timesteps, whereas on 4 nodes with 4 blocks $(1 \times 2 \times 2)$ and one processor per node the execution time is 18 seconds. As another example, we compare the subdivision into $1 \times 2 \times 2$ and $2 \times 4 \times 2$ blocks, both running on 4 nodes. In the first case, each node deals with 1 block, and in the second case each node does the computations for 4 blocks, and uses 2 processors for each block. So in that case the distributed memory model is adopted also *within* each single node. It appears that the latter case has a shorter execution time. It may be clear that some restructuring of the code is necessary in order to obtain reasonable performance. This will be the subject of another paper [16].

4.3 Optimization for cache-machines

In many parallel machines the processors use a hierarchical memory structure, consisting of a small amount of memory with a short access time (the cache) and a large amount of main memory with much longer access time. This long access time is the main reason why the performance of these machines is way below their (often impressive) peak. In the implementation of a numerical algorithm, it is essential to use the cache efficiently. Therefore, the number of load and store operations should be kept to a minimum, and quantities which are loaded from main memory should be reused as much as possible before being restored. Further, since elements from main memory are loaded into cache in chunks of a few consecutive elements, do-loops should be arranged such that main memory is traversed linearly (as is also necessary for efficient use of traditional vector-processors). Moreover, it will enable software-pipelining on RISC-processors, resulting in substantially faster execution.

To illustrate this, we compare two different ways to calculate the viscous flux. In the first method (method A) the various derivatives of the velocity fields and the temperature are calculated *consecutively*, and the viscous stress tensor and viscous heat flux are assembled and stored. Then the outer derivatives of the viscous flux are calculated, again consecutively. The resulting code is very well vectorizable and consists of very simple do-loops. In the second method (method B), we use the following observation. In the calculation of the derivatives, some averages can be used to contribute to various derivatives. Moreover, for all derivatives, the averaging weights in one direction are equal. Therefore we calculate all inner derivatives simultaneously, which also has the advantage that e.g. a vector u_1 needs to be loaded only once for the calculation of all its derivatives. An analogous fact holds for the weights. Further, the derivatives are not stored, but directly used to assemble the stress tensor and the heat flux. After that, all outer derivatives are calculated simultaneously. This results in about 30% less floating point operations, and substantially less load and store operations, resulting in better memory-performance. The drawback is the occurrence of (much) more complicated do-loop bodies, which puts a severe demand on the compiler in order to obtain suitable pipelining. It appears that on the T3d and the Paragon there is hardly any performance gain, and the performance is only about 20 % of peak. On one R8000 processor in the SGI Power Challenge (coupled to 4 MBytes of cache), the CPU-time of method B is half that of method A, with a performance of about 37 % of peak (110 Mflops). More details are to be found in ref. [17].

Acknowledgement

The time for the computations on the T3d was provided by the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF), which is financially supported by the Nederlandse Organisatie van Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO). One of the authors (HK) thanks the Institute for Fluid Dynamics at ETH Zürich for its hospitality during his stay there. The use of the Paragon has been made possible by courtesy of Prof. B. Müller, ETH Zürich. By courtesy of Silicon Graphics Inc. we were able to use the European Power Challenge Array of the SGI SuperComputer Technologies Centre in Cortaillod. We would like to thank Ruud van der Pas (SGI) for his assistance.

References

[1] B. Vreman, B. Geurts, H. Kuerten, J. Broeze, B. Wasistho and M. Streng, "Dynamic subgrid-scale models for LES of transitional and turbulent compressible flow in 3-D shear layers," Turbulent Shear Flow, (1995)

[2] A.Favre, Physics of Fluids, **26**, 2851, (1983)

[3] M. Germano, "Turbulence: the filtering approach," J. Fluid Mech. **238**, 325 (1992).

[4] B. Vreman, B. Geurts and H. Kuerten, "Realizability conditions for the turbulent stress tensor in large-eddy simulation," J. Fluid Mech. **278**, 351 (1994).

[5] A.W. Vreman, B.J. Geurts and J.G.M. Kuerten, "Subgrid-modelling in LES of compressible flows," Direct and Large-Eddy Simulation I, P.R. Voke, L. Kleiser and J.P. Chollet (editors), Kluwer, 133 (1994).

[6] U. Piomelli, T.A. Zang, C.G. Speziale and M.Y. Hussaini, "On the large-eddy simulation of transitional wall-bounded flows," Phys. Fluids A **2**, 257 (1990).

[7] J. Bardina, J.H. Ferziger and W.C. Reynolds, "Improved turbulence models based on LES of homogeneous incompressible turbulent flows," Department of Mechanical Engineering, Report No. TF-19, Stanford (1984).

[8] R.A. Clark, J.H. Ferziger and W.C. Reynolds, "Evaluation of subgrid-scale models using an accurately simulated turbulent flow," J. Fluid Mech. **91**, 1 (1979).

[9] S. Liu, C. Meneveau and J. Katz, "On the properties of similarity subgrid-scale models as deduced from measurements in a turbulent jet," J. Fluid Mech. **275**, 83 (1994).

[10] B. Vreman, B. Geurts and H. Kuerten, "On the formulation of the dynamic mixed subgrid-scale model," Phys. Fluids **6**, 4057 (1994).

[11] B. Vreman, B. Geurts and H. Kuerten, "Large Eddy Simulation of the temporal mixing layer using the Clark model," Memorandum No. 1213, University of Twente (1994).

[12] M.M. Rai and P. Moin, "Direct numerical simulation of transition and turbulence in a spatially evolving boundary layer," J. Comp. Phys. **109**, 169 (1993).

[13] B. Vreman, H. Kuerten and B. Geurts, "Shocks in direct numerical simulation of the confined three-dimensional mixing layer", Physics of Fluids, to appear (1995).

[14] J. Broeze, B. Geurts, H. Kuerten and M. Streng, "Multigrid acceleration of time-accurate DNS of compressible turbulent flow," Copper Mountain (1995).

[15] E.F. van de Velde, "Concurrent Scientific Computing," Springer Verlag, New York (1994).

[16] M. Streng and R. van der Pas, "Implementation of a compressible flow solver on the Power Challenge Array", in preparation.

[17] M. Streng and R. van der Pas, "Some performance considerations for the R8000 Microprocessor", in preparation.