ELSEVIER

Simulation Practice and Theory 6 (1998) 119–147

SIMULATION PRACTICE ≡ THEORY

The effectiveness of domain balancing strategies on workstation clusters demonstrated by viscous flow problems

Martin Streng [a,1], Eric (H.H.) ten Cate [c,*], Bernard J. Geurts [b], Hans (J.G.M.) Kuerten [b]

[a] *Hollandse Signaalapparaten BV, Zuidelijke Havenweg 40, 7554 RR Hengelo, The Netherlands*
[b] *Faculty of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands*
[c] *Department of Ministry of Transport, Public Works and Water Management, National Institute for Coastal and Marine Management (RIKZ), P.O. Box 20907, 2500 EX The Hague, The Netherlands*

Received 3 January 1997; revised 18 September 1997

Abstract

We consider several aspects of efficient numerical simulation of viscous compressible flow on both homogeneous and heterogeneous workstation-clusters. We consider dedicated systems, as well as clusters operating in a multi-user environment. For dedicated homogeneous clusters, we show that with respect to the turn-around time, there is an optimal number of workstations to solve a given flow-problem. This number depends on the actual implementation of the solver. For non-dedicated heterogeneous clusters we apply dynamic domain-balancing techniques in order to obtain an optimal turn-around time *for the flow-simulation job only*, taking into account the activities on the cluster arising from the other applications. We show that the decision which technique should be used depends on various aspects, such as the character of the load-fluctuations arising from the other applications, whether the flow-simulation job is computationally intensive, and the underlying hardware. The effects of these aspects on this decision are analyzed. Although it can be concluded that no domain-balancing technique is the best under *arbitrary* circumstances, such an analysis may guide the decision for a *particular* load-situation. Moreover, we indicate how such a decision can be supported by a comparison of the various balancing strategies using a simulation study. This is illustrated for a specific load-situation. © 1998 Published by Elsevier Science B.V.

Keywords: Dynamic load-balancing; Parallel computing; Distributed computing; Workstation clusters; Domain-decomposition; Navier-Stokes equations

* Corresponding author. E-mail: h.h.tcate@rikz.rws.minvenw.nl.

[1] M. Streng was supported by NWO-grant nr. 610-02-100.

PII S0928-4869(97)00035-9

1. Introduction

One of the most important developments in high-performance computation of flow-simulations is the efficient use of parallel platforms. Although nowadays powerful parallel machines become available, they are still rather expensive. In many production environments, however, a cluster of workstations is present, which in principle can serve as a parallel platform. The advantage of this approach is that high-performance computing is possible at very low cost. However, besides the fact that parallel programming is, for many applications, often more involved on a cluster than on an integrated parallel machine, there are three other problems that should be addressed. Firstly, since parallelism can only be achieved by message-passing, messages must be sent over the network, which is a slow process. Hence the speed-up may be affected. Secondly, most existing clusters are heterogeneous, i.e. not all workstations in the cluster have the same capacity. This unbalance should be properly taken into account in order to reduce the turn-around time of a given application. In the third place, (some of) the workstations may simultaneously be used for other applications, and in most cases it will not be possible to schedule all jobs arising from the various other activities in such a way that the cluster can be made dedicated for a long-running parallel job.

The effects of these 'disadvantages' on the performance of an application depend on the algorithmic details of the application. In this paper we consider several strategies aimed at a reduction of the turn-around time for time-accurate numerical simulation of compressible viscous flow. However, the results of these investigations can be used for several other applications as well. The parallelization of such flow-simulations is usually achieved by domain-decomposition techniques, resulting in block-structured algorithms. In these algorithms, each workstation or processing element (PE) advances the solution in one block of the flow-domain over one time-step. For the computation of the solution in the next time-step, information is needed which is allocated on processors dealing with neighboring blocks in the flow-domain. Therefore, essentially a global synchronization takes place every time-step, which implies a certain amount of communication between the PE's. In large-scale applications the computations for one block are quite time-consuming, and the communication can be 'hidden' behind these computations, i.e. scheduled in the algorithm such that the computations are not interfered. In the case of a dedicated cluster, as long as the network is not saturated, the only overhead arising from the parallelization is ideally due to the encoding and decoding of the messages. Depending on the speed of the network, the speed of the PE's and the amount of memory per workstation, bounds can be given on the maximal attainable speed-up, and the optimal number of work stations to be used for the simulation of a given flow-problem. In an example it is shown that this number, as well as the maximal attainable speed-up and the minimal turn-around time depend strongly on the actual implementation of the spatial discretization of the flow equations.

The progress of a flow-simulation with a block-structured parallel flow-solver depends on the differences in the size of the individual blocks in relation to the speed of the PE's on which they are allocated. Because of the global synchronization

step, the calculation-time is dominated by the speed of the slowest PE. The ability of the PE's to contribute to the flow-simulation can be different due to the inhomogeneity of the cluster, but also because a PE may be involved in the processing of other jobs. Especially in a parallel production environment both effects will almost inevitably occur in practical situations, where a production run will make use of a number of PE's and still can last for several (many) hours. Therefore, the most desirable situation would be that slower PE's are assigned less work (smaller blocks) than faster ones, and that most of the work in the flow-simulation is done on PE's which are least claimed by other jobs. In order to achieve this situation, some form of dynamic load-balancing should be employed.

Dynamic load-balancing has been discussed in literature quite frequently. We will only refer to the following restrictive list of review-like papers: Refs. [1–5]. Most authors are mainly concerned either with the optimal redistribution of *all* running jobs on a multicomputer, or with migration of data associated with one particular job on a dedicated system. Here, we consider the redistribution of work associated with one particular job on a non-dedicated heterogeneous cluster. This is considerably more complex than the cases mentioned above, since the interaction of the load-balancing of our job with the other tasks has to be taken into account, in order to judge the effectivity of the method.

We modify and extend three existing load-balancing algorithms for this case, viz. the diffusion method (cf. Ref. [6]), the generalized dimension-exchange method (cf. Ref. [7]), and a multilevel method (cf. Ref. [8]). Moreover, we present a new family of global redistribution methods, and indicate the similarities between all these strategies.

The decision which technique should be preferred depends much on the character of the load-fluctuations, on whether the flow-simulation job is computationally intensive, and on the underlying hardware. Therefore, no single domain-balancing technique is the best under *arbitrary* circumstances. For any *particular* load-situation, an educated guess has to be made. In order to guide such a guess, we analyze the influence of the above mentioned effects on the performance of a balancing strategy and we show that the choice for one of the strategies can be supported by a simulation study. In such a study the performance of the CFD-computations resulting from the balancing strategy in combination with the CFD-algorithm, the underlying hardware and the presumed load-situation is simulated. In this way the most suitable balancing strategy can be determined without actually performing a real (time-consuming) flow-simulation. This is illustrated by a comparative study of the various techniques for a special load-situation.

As a side result, we show that the characteristics of the diffusion algorithm can be extended to massively parallel systems. In this way we obtain a description of the behavior of this algorithm on large clusters or MPP systems. It turns out that this behavior is described by a partial differential equation of the Fokker–Planck type, in which the unknown is the load per processor on every time-step.

The set-up of the paper roughly follows these lines. In Section 2, a representative simple flow problem is presented. In Section 3 an algorithm for the parallel solution

of this problem is given. We deal with dedicated clusters in Section 4, and consider the optimal number of workstations to solve a given problem, both for our model problem and for the solution of the 3d Navier-Stokes equations. Moreover, we determine the optimal block-sizes in a domain-decomposition. The last two sections are devoted to non-dedicated clusters. In Section 5 the various balancing algorithms are presented. In Section 6 the effectiveness of these algorithms is discussed.

Because many of our considerations in this paper are also applicable for distributed systems other than workstation-clusters, we will use the term processing element (PE) rather than workstation. We will use the notational convention that quantities which are considered to belong to $\mathbb{R}^n$ for some n are denoted in boldface. The components of such a quantity are denoted by the same symbol, but in roman, with an index. So if e.g. $\boldsymbol{x} \in \mathbb{R}^2$, then $\boldsymbol{x}=(x_1, x_2)$. The symbols $\boldsymbol{x}_i$ and $\boldsymbol{x}_{ij}$ represent indexed vectors.

2. A prototype flow problem

In this section we introduce — for future reference — some basic concepts through a detailed definition of a simple flow problem. Let $A=(0, 1)\times(0, 1)$, $I=[0, \infty)$, and u: $I\times A\to\mathbb{R}$. A point in $I\times A$ is denoted by (t, x_1, x_2), or $(t, \boldsymbol{x})$. As a model problem we consider the 2d scalar Burgers equation on $I\times A$:

$$\frac{\partial u}{\partial t}+\frac{1}{2}\frac{\partial u^2}{\partial x_1}+\frac{\partial u}{\partial x_2}=\mu\Delta u, \tag{1}$$

with $\mu\in(0, \infty)$, and with boundary conditions $u(t, 0, x_2)=3/2$, $u(t, 1, x_2)=-1/2$, $u(t, x_1, 0)=3/2-2x_1$ and $\partial u/\partial x_2\,(t, x_1, 1)=0$. Moreover, there is an initial condition $u(0, x_1, x_2)=3/2-2x_1$.

In order to solve the system (1) numerically, given initial and boundary conditions, we endow our domain A with an orthogonal grid, with mesh-sizes h_1, h_2 (such that $1/h_1\equiv N_1\in\mathbb{N}$, and $1/h_2\equiv N_2\in\mathbb{N}$). The grid is denoted by $\tilde{A}$, and the points in $\tilde{A}$ by $\boldsymbol{x}_{ij}$:

$$\tilde{A}=\{\boldsymbol{x}_{ij}\in\mathbb{R}^2 | \boldsymbol{x}_{ij}=(ih_1, jh_2) \text{ for some } i, j \text{ with } 0\le i\le N_1,\ 0\le j\le N_2\},$$

For $i=1, \ldots, N_1$, Let $\tilde{C}_i$ denote the set given by

$$\tilde{C}_i=\{\boldsymbol{x}_{ij}\in\tilde{A} | j=0, \ldots, N_2\}.$$

This set is called the *i-th column* of $\tilde{A}$. For a function u: $I\times A\to\mathbb{R}$ we will shorthand

$$u_{ij}(t)=u(t, \boldsymbol{x}_{ij}),$$

and we will often omit the parameter t for the sake of brevity.

Using a second order accurate finite volume spatial discretization on a 3×3 stencil, and following the method of lines, we end up with a discrete version of our

original problem, which is a system of ordinary differential equations

$$\frac{\mathrm{d}}{\mathrm{d}t} u_{ij}(t) = \Phi_{ij}, \quad i=1, \ldots, N_1-1, j=1, \ldots, N_2-1, \tag{2}$$

where Φ_{ij} represents the discretized flux, which depends on the values of u_{ij}, $u_{i\pm1,j}$, $u_{i,j\pm1}$ and $u_{i\pm1,j\pm1}$ (cf. Ref. [9]). The values $u_{0j}(t)$, $u_{N_1 j}(t)$ and $u_{i0}(t)$ are determined by the discretization of the boundary-conditions, and $u_{i_2}(t)$ is obtained by linear extrapolation. The u_{ij} fulfill an initial condition which results from the discretization of the original initial condition.

Eq. (2) is integrated with an explicit Runge–Kutta (RK) method. If the time-step is denoted by Δt, we consider s-stage RK integration methods of the following form:

$u_{ij}^0 = u_{ij}(0)$, and for $n=1, 2, \ldots,$

$n_{ij}^{n,0} = u_{ij}(n\Delta t)$,

$n_{ij}^{n,k} = u_{ij}(n\Delta t) + a_k \Delta t \Phi_{ij}^{n,k-1}, \quad k=1, \ldots, s,$

$u_{ij}((n+1)\Delta t) = u_{ij}^{n,s}$,

for some coefficients a_k, $k=1, \ldots, s$. Introducing the *time-level* $\tau = sn+k-1$ and (by abuse of notation) denoting $u_{ij}^{n,k}$ by u_{ij}^{τ}, this takes the general form

$$u_{ij}^{\tau+1} = \mathscr{F}(u_{ij}^{\tau}, u_{i\pm1,j}^{\tau}, u_{i,j\pm1}^{\tau} u_{i\pm1,j\pm1}^{\tau}, u_{ij}^{\tau \operatorname{div} s}),$$

where $\tau \operatorname{div} s = sn + k \operatorname{div} s = sn$.

3. Parallel processing of the prototype problem

In this section we discuss an algorithm for solving the prototype problem on a distributed memory parallel computer, using a simple domain-decomposition.

If the prototype problem is to be solved on a parallel computer, we have to distribute the work over the PE's. Distributing the work will, in this paper, be considered equivalent to distributing the data. We will use the convention that if we speak about allocating part of the grid to a PE, we imply that the necessary storage for all associated quantities (e.g. the flow field and the fluxes) is allocated on that PE as well. In principle, the calculation of $u_{ij}^{\tau+1}$ out of u_{ij}^{τ} for a given time-level τ and one pair (i, j) can be regarded as one unit of work. However, this calculation uses e.g. the values $u_{i\pm1,j\pm1}^{\tau}$ for the computation of the discrete flux Φ_{ij}. Therefore, distributing the data u_{ij} arbitrarily over all the PE's would lead to excessive communication overhead.

It is better to allocate a contiguous block of the grid on each PE. Then only the calculation of Φ_{ij} for x_{ij} on the boundaries of the blocks requires communication. For a square 2d grid, the blocks which have the smallest ratio of boundary-points to interior points and still fill up the whole grid, are squares. However, in later sections, where we discuss non-dedicated clusters, we will have to redistribute the blocks over the PE's in such a way that not all blocks have equal size. Using the

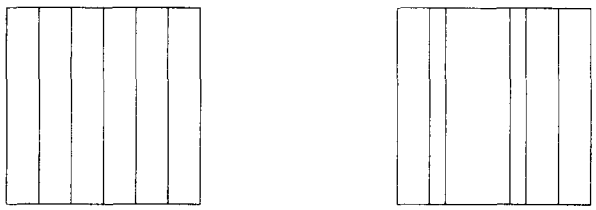

Fig. 1. Decomposition of a square computational space into equal squares (a) and slices (c). In the pictures (b) and (d), one block is resized.

decomposition into squares, this can lead to complicated neighborhood structures (cf. Fig. 1a, b). Moreover, both for vector processors and for most RISC-processors used in present day computers, the availability of long contiguous arrays containing the data corresponding to one spatial dimension of the physical problem is beneficial for the performance, due to the mechanisms of vector- or software pipelining. For these reasons we will consider a decomposition into slices. In this decomposition, which is also used by e.g. Ferraro et al. [10] for a different physical problem, resizing of the blocks will result in an unaltered neighborhood structure (cf. Fig. 1c, d). Note that, in contrast with the situation occurring in the context of unstructured grids, the neighborhood structure of this problem is quite obvious and has been chosen by simple arguments. We next describe this structure in detail.

For n_-, $n_+ \in \mathbb{N}$, let

$$\tilde{A}_{n_-,n_+} = \{x_{ij} \in \tilde{A} | n_- \leq i \leq n_+\}.$$

The points $x_{ij} \in \tilde{A}_{n_-,n_+}$ with $n_- < i < n_+$ are called the *bulk-points*, and those with $i = n_-$ or $i = n_+$ are the *dummy-points*. In the dummy-points, values to be received from the PE's dealing with the adjacent blocks are stored. Suppose we have P PE's. Let $n_-(p)$, $n_+(p)$, $p = 1, \ldots, P$ be natural numbers such that

$$n_-(1) = 0,$$

$$n_+(p) = n_-(p+1) + 1, \quad p = 1, \ldots, P-1,$$

$$n_+(P) = N_1,$$

$$n_+(p) - n_-(p) \geq 2, \quad p = 1, \ldots, P.$$

Then

$$\tilde{A} = \bigcup_{p=1,\ldots,P} \tilde{A}_{n_-(p),n_+(p)}.$$

We will frequently abbreviate

$$\tilde{A}_p = \tilde{A}_{n_-(p),n_+(p)}.$$

Note that

$$\tilde{A}_p \cap \tilde{A}_{p+1} = \tilde{C}_{n_+(p)} \cup \tilde{C}_{n_-(p+1)}.$$

Now we want PE p to perform the calculations for the solution of the differential equations for the bulk-points in $\tilde{A}_p$. Therefore we allocate each $\tilde{A}_p$ on one PE p. The columns $\tilde{C}_{n_+(p)}$, $p=1, \ldots, P-1$ and $\tilde{C}_{n_-(p)}$, $p=2, \ldots, P$ are allocated on two PE's. Given the values of u_{ij}^{τ} for $n_-(p) \le i \le n_+(p)$ and $0 \le j \le N_2$ for a given time-level τ, processor p can compute the discrete fluxes Φ_{ij} for $n_-(p) < i < n_+(p)$, and hence the values $u_{ij}^{\tau+1}$ on time-level $\tau+1$. In order to calculate $u_{ij}^{\tau+2}$ for $i=n_-(p)+1$ and $i=n_+(p)-1$, the values of $u_{ij}^{\tau+1}$ for $i=n_-(p)$ resp. $i=n_+(p)$ need to be transferred from processor $p-1$ resp. $p+1$. Note that (for $1<p<P$), $u_{n_+(p),j}=u_{n_-(p+1)+1,j}$, and $u_{n_-(p),j}=u_{n_+(p-1)-1,j}$. An exception occurs if $p=0$ resp. $p=P$, since then $u_{0,j}^{\tau+2}$ resp. $u_{N1,j}^{\tau+2}$ are determined by the boundary conditions.

Since we are using a distributed platform, we have to employ message-passing to transfer the dummy-points. If we use asynchronous (non-blocking) send operations and synchronous (blocking) receive operations, it is efficient to advance the solution for the columns $n_-(p)+1$ and $n_+(p)-1$ first, send the results to the neighboring PE's, and then continue with the other bulk-points of $\tilde{A}_p$. In this way, most of the communication can be hidden behind the computations. This is confirmed by Cap and Strumpen [11], where experiments are reported concerning a similar problem. So, all processors p, $p=1, \ldots, P$ will run the following algorithm:

$\tau=0$
Initialize u_{ij}^0, $n_-(p) \le i \le n_+(p)$, $0 \le j \le N_2$
repeat
Calculate $u_{n_-(p)+1,j}^{\tau+1}$, $0 \le j \le N_2$
if $p \neq 1$ then send $u_{n_-(p)+1,j}^{\tau+1}$, $0 \le j \le N_2$ to processor $p-1$
Calculate $u_{n_+(p)-1,j}^{\tau+1}$, $0 \le j \le N_2$
if $p \neq P$ then send $u_{n_+(p)-1,j}^{\tau+1}$, $0 \le j \le N_2$ to processor $p+1$
Calculate $u_{ij}^{\tau+1}$, $n_-(p)+1 < i < n_+(p)-1$, $0 \le j \le N_2$
If $p \neq 1$ then receive $u_{n_-(p),j}$, $0 \le j \le N_2$ from processor $p-1$
If $p \neq P$ then receive $u_{n_+(p),j}$, $0 \le j \le N_2$ from processor $p+1$
$\tau=\tau+1$
until some stop criterion.

For simplicity, we will allow $n_-(p)$, $n_+(p)$ to be real-valued, rather than integer in the rest of the paper. This greatly simplifies the analysis, and since in a real application $N_1 \gg P$, this is not unreasonable. In a real application, these values should be rounded.

4. Dedicated systems

In this section we consider the interplay of the CFD-problem size, the memory sizes and processor speeds of the workstations, and the bandwidth of the interconnection network, for dedicated systems. In the first subsection we will show in two examples that there is an optimal number of PE's given the hardware characteristics and the actual implementation of the discrete flux calculations. It is shown that the optimal implementation for this kind of platforms is different from the implementation which is traditionally used for vector-computers. To illustrate the point it is

sufficient to consider homogeneous clusters. In the second subsection the optimal block-decomposition for heterogeneous clusters is derived.

4.1. The optimal number of workstations

In this subsection we will assume for simplicity that the workstation-cluster is homogeneous and dedicated, and consists of P PE's, each with M words of memory and a sustained speed of S floating-point operations per second (Flops). Further, the network connecting these PE's is shared by all PE's and has a sustained bandwidth of B words per second.

In particular, we wish to determine the optimal number of PE's to solve a given CFD-problem. In order to motivate this, consider e.g. Burgers' equation of Section 2. Intuitively, one expects that with a few PE's this problem can be solved in less time than by using only one. However, if the number of PE's is too large, the problem is decomposed into too many tiny pieces, so the network-traffic due to the communication between the PE's may be so large that the network becomes saturated. The optimal situation hence is obtained as a 'balance' between these effects.

We will consider two examples. The first deals with Burgers' equation of Section 2. In this example the memory of the workstations is not considered a critical issue. The second example is devoted to integration of the three-dimensional Navier–Stokes equations for direct numerical simulation of compressible viscous flow, where a fourth order spatial discretization is used. Here the amount of memory of the workstations is a critical issue, which has consequences for the implementation of the algorithm. In these examples we assume that the grid is sufficiently fine, i.e. $N_1, N_2 \gg 1$. In the Navier–Stokes case, the same is assumed to hold for the number N_3 of grid cells in the third coordinate direction, i.e. $N_3 \gg 1$. Moreover we assume that the number of PE's, P, is much smaller than N_1.

4.1.1. Example 1

Suppose we want to compute the solution to Burgers' equation on a grid containing $N_1 \times N_2$ grid-cells, using the time-stepping method and spatial discretization as mentioned in Section 2. We assume that we need f floating-point operations in order to compute $u_{ij}^{\tau+1}$ out of u_{ij}^{τ}. Further, we assume that each PE deals with an equally sized part of the flow-domain. Then, if we use P PE's, for each Runge–Kutta stage, the time spent in computation T_{flop} is, to a good approximation, given by

$$T_{\text{flop}} = \frac{N_1 N_2 f}{SP}.$$

The communication time is the sum of the times needed for encoding and packing the data, the time necessary for the actual transmission of the data and the time needed for decoding and unpacking the data after receiving. For the partition of the flow-domain as described in the previous section, each PE has to send approximately $2N_2$ words of data, except for the PE's dealing with the first and the last block. The packing and unpacking of the data can be done in parallel, and is assumed to be

possible in a negligible amount of time. The total transmission time T_{trans}, is given by

$$T_{\text{trans}} = \frac{2(P-1)N_2}{B}.$$

Using asynchronous send operations, the transmission can be done during the computation if $T_{\text{trans}} < T_{\text{flop}}$. This leads to

$$\frac{2(P-1)N_2}{B} < \frac{N_1 N_2 f}{SP} \Rightarrow P < P^* = \frac{1}{2} + \frac{1}{2}\sqrt{1+2BN_1 f/S}.$$

This implies that the total turn-around time for the calculation of one stage is

$$\frac{N_1 N_2 f}{SP} \quad \text{if } P \leq P^*$$

$$\frac{2(P-1)N_2}{B} \quad \text{if } P > P^*$$

The smallest turn-around time occurs for $P = \text{rnd}(P^*)$ PE's, where $\text{rnd}(x)$ denotes the value of x rounded to the nearest integer. A typical situation is the case where $N_1 = N_2 = 300$, $f = 40$, $S = 10 \times 10^6$, $B = 0.15 \times 10^6$. Then $\text{rnd}(P^*) = 10$ (cf. Fig. 2). Note that in the limit for vanishing bandwidth B or infinite computing speed S, $P^* \to 1$.

4.1.2. Example 2

The second example concerns the direct numerical simulation of 3d compressible viscous flow. In this case, instead of Burgers' equation, the full Navier–Stokes equations are integrated, using the same Runge–Kutta time-stepping algorithm as described in Section 2, and with a $5 \times 5 \times 5$ stencil for the (fourth order) spatial

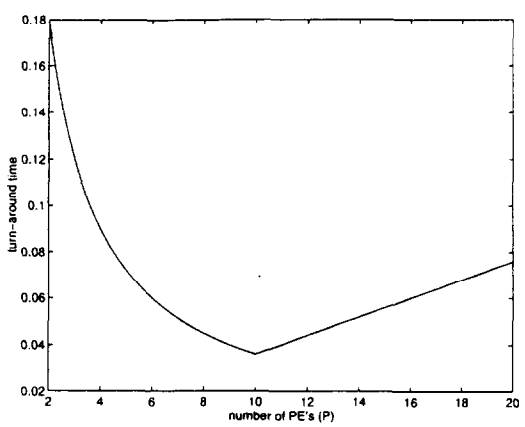

Fig. 2. Turn-around time for parallel execution on P PE's of one Runge–Kutta stage for Burgers' equation.

discretization. In this case the analogue of a column is a *slab*, which we denote by the same symbol:

$$\tilde{C}_i = \{\boldsymbol{x}_{ijk} | 0 \le j \le N_2,\ 0 \le k \le N_3\}.$$

Assume that we have a block-shaped grid with $N_1 \times N_2 \times N_3$ grid-cells, which we partition into slabs of each $(N_1/P) \times N_2 \times N_3$ cells. Then, using the same notation as in the previous example, at each Runge–Kutta stage 2 layers of dummy-points per processor must be transferred to each of its neighbors. Note that the processors dealing with the first and the last slab only have one neighbor. Per dummy-point there are five words to transmit, corresponding to the five dependent variables, e.g. mass-density, energy and the 3d momentum vector. So the total network-traffic for each stage is approximately

$$(P-1) \times 2 \times 2 \times 5 \times N_2 \times N_3 \text{ words.}$$

The total computation time is

$$T_{\text{flop}} = \frac{N_1 N_2 N_3 f}{SP}.$$

Hence communication will be the bottleneck if

$$\frac{20(P-1)N_2N_3}{B} > \frac{N_1N_2N_3f}{SP} \Rightarrow P > P^* = \frac{1}{2} + \frac{1}{2}\sqrt{1 + \frac{N_1 fB}{5S}}.$$

Again, the turn-around time is optimal if $P = \text{rnd}(P^*)$. In this example however, it is not realistic to ignore the memory of the PE's. Suppose we have an algorithm which needs $mN_1N_2N_3$ words of memory. In practice, m will be at least 10, since we need 5 points per grid-point to store the flow-quantities, and we must keep u^{τ}_{ijk} as well as $u^{\tau \text{ div } s}_{ijk}$. Usually, the number f depends on m, since often temporary storage can be used to decrease this number by reuse of calculated quantities. Therefore, we use the notation f_m to express this dependency.

If each PE has M words of memory, we need at least

$$P_{\min} = \left\lceil \frac{mN_1N_2N_3}{M - 4N_2N_3} \right\rceil$$

workstations to solve the flow-problem, where $\lceil x \rceil$ denotes the smallest integer larger than x. The term $4N_2N_3$ arises from the storage needed for the dummy-points of each block. So if $P_{\min} > P^*$, the turn-around time for a Runge–Kutta stage is dominated by the communication.

We will illustrate this for two practical cases. In both cases we use $N_1 = N_2 = N_3 = 256$, $S = 25 \times 10^6$ flops, $B = 0.15 \times 10^6$ words per second, $M = 32 \times 10^6$ word. The main difference between these cases lies in the implementation of the flux-calculation.

Case 1. This is the typical vector-computer implementation. Now the flux is not calculated for each grid-point individually, but is instead generated in several sweeps

through the whole part of the flow-domain which is allocated on one PE, where in every sweep a partial flux for all grid-points is calculated. The total flux is then some accumulation of these partial fluxes. A common property of this type of implementation is a rather large amount of temporary storage and a fairly low number of floating-point operations. A typical case is $m=30$, $f_m=50$. Then $\mathrm{rnd}(P^*)=4$ and $P_{\min}=16$, so the total turn-around time for one Runge–Kutta stage is (neglecting encoding and decoding times) 131 s. In this case the turn-around time is dominated by the communication.

Case 2. In this case the flux-calculations are done for each grid-point individually, resulting in modest storage-requirements, but also in inefficient vector-code. However, for parallel computing this method is in some cases favorable. Assume $m=15$, $f_m=500$. Then $\mathrm{rnd}(P^*)=10$, $P_{\min}=8$, and the turn-around time for one Runge–Kutta stage is 84 s.

The reason that the second case has a smaller turn-around time is roughly speaking that — to a certain extent — extra floating-point operations are cheaper than communication. This implies that the traditional trade-off between memory usage and number of floating point operations is for parallel computing to be extended to a trade-off between memory usage, number of floating point operations and communication.

4.2. *Optimal block-sizes for heterogeneous clusters*

In this subsection we show how to determine the optimal block-sizes for a heterogeneous cluster. In literature this is usually referred to as static load-balancing. Again we take our prototype flow-problem as a case study. We assume that the communication can be hidden behind the computation (i.e. $P<P^*$ in the case of a homogeneous cluster, cf. also Remark 4.1). First we introduce some notation.

We define T_p as the time needed by PE p to compute $u_{ij}^{\tau+1}$ out of u_{ij}^{τ} for *all* bulk-points $\boldsymbol{x}_{ij}\in\tilde{A}_{n_-(p),n_+(p)}$. Let α_p be the time needed by PE p to compute $u_{ij}^{\tau+1}$ out of u_{ij}^{τ} for *one column* $\tilde{C}_i$. Further, let $X_p=n_+(p)-n_-(p)-1$. Then

$$T_p=\alpha_p X_p.$$

The total turn-around time T_{total} for one time-level is then

$$T_{\text{total}}=\max_p T_p.$$

In order to find the optimal block-sizes, we should solve the following optimization problem

$$\min_{X_1,\ldots,X_P}\ \max_p \alpha_p X_p \quad \text{subject to} \sum_{p=1}^{P} X_p=N_1$$

It can be proved that the solution of this problem equals the solution of the system

$$\alpha_p X_p=\alpha_{p+1}X_{p+1},\quad p=1,\ \ldots,\ P-1,\qquad \sum_{p=1}^{P} X_p=N_1,$$

which is intuitively clear, since now none of the PE's will be idle. With the notation

$$\frac{1}{A} = \frac{1}{\alpha_1} + \ldots + \frac{1}{\alpha_P},$$

this leads to

$$X_p = N_1 \frac{A}{\alpha_p}. \tag{3}$$

Remark 4.1. The optimal time needed to compute one RK-stage for the whole flow-domain is $N_1 A$. If there are P PE's, the communication can be hidden behind the computation if

$$N_1 A > \frac{2(P-1)N_2}{B}.$$

Remark 4.2. We observe a striking analogy with the situation occurring in the theory of electrical networks. Consider a circuit consisting of P parallel resistances $R_1, \ldots, R_P$, which carries a total current I. The total resistance R of the circuit satisfies

$$\frac{1}{R} = \frac{1}{R_1} + \ldots + \frac{1}{R_P},$$

and the current I_p through resistor p is given by

$$I_p = \frac{IR}{R_p}.$$

So the analogy is that X_p corresponds with the current through resistor p, α_p with the resistance of resistor p, and A corresponds with the total resistance of the circuit. Further, T_p corresponds with the voltage over R_p, which is equal for all p in the steady-state.

Remark 4.3. Note that we have obtained the optimal block-sizes given the shape of the blocks (i.e. given the communication structure of the parallel algorithm). This is much more complicated in the context of unstructured grids, where the determination of the best communication structure is part of the balancing problem. This is also a little more complicated in the case of extra work generation by means of grid refinement for example, which is briefly considered in Section 5.6. Then N_1 and N_2 vary in time, which makes the optimization problem more difficult.

5. Non-dedicated systems

5.1. Introduction

In this section and the following we consider non-dedicated clusters. In this situation, the CFD job we are running will be called *our job*, and besides our job

there are some *other tasks* running on the cluster. We are interested in ways to ensure that the turn-around time for our job is minimal, taking into account the work that has to be done on the cluster for the other tasks. This can be done by allocating smaller blocks on PE's which are heavily loaded and/or slow, and bigger blocks on PE's which are relatively unloaded and/or fast. Because the load on the PE's changes in time, this allocation has to be renewed frequently. In our particular case, we will consider a reallocation of the domain after each Runge–Kutta stage.

For this purpose we consider dynamic load-balancing strategies. Dynamic load-balancing has been mentioned in literature quite frequently. We will only refer to the following restrictive list of review-like papers: Refs. [1–5]. Most authors are concerned with either one of the following two cases:

(i) Distributing all tasks in the cluster such that the cluster as a whole is used optimally. This especially occurs in the context of distributed operating systems. See Schabernack [2] for a survey.

(ii) Distributing the work of one particular job optimally over all PE's in a cluster which is dedicated to that job. In the context of CFD-computations, this is usually related to adaptive unstructured meshes (cf. e.g. Williams [4]).

In this paper we will consider the case where the cluster is not dedicated, but one also does not want to install some form of a distributed operating system. There are several reasons for not installing a distributed operating system. In a production environment, many other applications are running, which possibly make use of the existing operating system, or are licensed to one particular machine. Further, using all the features of a distributed operating system may decrease the portability of the software. Finally, the transition to a new operating system is usually quite involved, and affects all users of the cluster.

These assumptions are quite realistic for many production environments. However, our aim is considerably more complex than the cases mentioned above, since now we have to take the interaction of the load-balancing of our job with the other tasks into account, in order to judge the effectivity of the method. Some related work in this direction, from the viewpoint of developing data-parallel programming languages, has recently been undertaken by Nedeljkovic and Quinn [12].

Because only the work/data associated with our job is reallocated, we will speak about *domain-balancing* in order to avoid confusion. Of course domain-balancing is closely related to load-balancing in the traditional sense. In particular, various load-balancing techniques can be modified to domain-balancing. In literature several types of load-balancing strategies are mentioned. Roughly speaking, we distinguish between local and global strategies. In local strategies there is only migration of work between neighboring PE's. We refer to Willebeek–Lemair and Reeves [1] for a survey of local strategies. The most well-known are diffusive methods (cf. the classical paper by Cybenko [6]) and dimension exchange methods (cf. e.g. Xu and Lau [7]). Local strategies are fully distributed in the sense that each PE decides itself, based on information from adjacent PE's only, whether or not to participate in the process of load-balancing. Usually, these strategies require less synchronization than global methods, but they react slowly on variable load. It will appear, however, that in the presence of rapidly fluctuating loads, this can be a good property. In

global strategies the load-characteristics of all PE's are collected, and a new distribution of the data is calculated. These methods can react fast on load-variations, but require much stricter synchronization than local methods. Well-known methods of this kind are e.g. the various spectral bisection methods, cf. e.g. the paper by Williams [4], and the gradient model algorithms, as described by e.g. Lin and Keller [13]. We will study one particular type of global methods already known from literature, namely the multilevel methods. In these methods the available PE's are split into a hierarchy of sub-clusters. Balancing is done between sub-clusters which have the same level in the hierarchy, see e.g. the paper by Horton [8].

Since we want to balance the load of one particular job (viz. our job) on a non-dedicated system, we cannot directly apply the methods described above. However, for representative methods from each of these categories, we show that it is possible to modify them in an appropriate way. Moreover, we will give a new global method, based on the fact that the static domain-balancing can be achieved very easily.

In order to judge the effectiveness of these methods, we have to consider the interaction with the other tasks. We are interested in a heuristic indicating which type of method should be used given the characteristics of the other tasks, in order that the turn-around time for our job is minimal. In this respect, there are two effects that should be taken into account.

(i) Due to a domain-imbalance the turn-around time is larger than optimal, since some of the PE's cannot advance the solution of their part of the flow problem further, because they are waiting for the dummy-points from their neighboring PE's.

(ii) Due to a redistribution of the work (in order to obtain a balanced domain), we have to make some communication costs, which slows down the execution of our job.

In fact, at each domain-balancing step, it should be decided whether it is (in the long run) better to balance or not, and, if appropriate, which method should be used. This, however, is impossible to do optimally, since the load due to the other tasks may be changing in an unpredictable way.

Hence, the decision which method should be used depends on the (statistical) distribution of the other tasks. But even then it is mathematically very hard, if not impossible, to predict the expected turn-around time of our job, given this distribution and given a load-balancing method. In this paper we confine ourselves to a model study. This will show that there is no method which performs best under all circumstances. Moreover, we will obtain an understanding of the behavior of the strategies in relation to the varying load of the other tasks, and provide heuristics to guide the decision for application of one of the strategies. It is indicated that, for a given load-situation, these heuristics can be confirmed by a simulation of the progress of a CFD-calculation in combination with the balancing-method in the presence of the particular load-situation.

Before presenting the various dynamic domain-balancing strategies, we introduce some notation. Because $n\ (p)$, $n_+(p)$ are assumed to vary every time-level, we denote them by $n^\tau_-(p)$ and $n^\tau_+(p)$, $p=1, \ldots, P$, $\tau=0, 1, 2, \ldots$. For $p=1, \ldots, P$, let $X^\tau_p=n^\tau_+(p)-n^\tau_-(p)-1$, and let γ_p be the time needed by workstation p to calculate $u^{\tau+1}_{ij}$ out of u^τ_{ij} for one column $\tilde{C}_i$ in a situation where PE p is dedicated to this task,

i.e. there are no other jobs running on this workstation. In general, in a heterogeneous cluster, γ_p will be different for each p. The number of other jobs which are running on workstation p at time-level τ is denoted by l_p^τ. Then, because of timesharing, the real time needed by workstation p to calculate $u_{ij}^{\tau+1}$ out of u_{ij}^τ for one column $\tilde{C}_i$ is given by

$$T_p^\tau = \gamma_p(1+l_p^\tau)X_p^\tau,$$

where we assumed a times-lice which is small compared to $\gamma_p X_p^\tau$. We put

$$\alpha_p^\tau = \gamma_p(1+l_p^\tau),$$

which reduces to the definition given in Section 4.2 if $l_p = 0$.

In all strategies given below we determine $X_p^{\tau+1}$, $p=1, \ldots, P$ as the solution or an approximation to the solution of the system

$$\begin{aligned} &\alpha_p^\tau X_p^{\tau+1} = \alpha_{p+1}^\tau X_{p+1}^{\tau+1}, \quad p=1, \ldots, P-1 \\ &\sum_{p=1}^{P} X_p^{\tau+1} = \sum_{p=1}^{P} X_p^\tau, \end{aligned} \tag{4}$$

where $\alpha_p^\tau = T_p^\tau / X_p^\tau$, for $p=1, \ldots, P$. We explicitly solve these equations for the case $P=2$, and for future reference express the results somewhat more generally. We obtain after some manipulations (in our case $p=1$)

$$\begin{aligned} X_p^{\tau+1} &= X_p^\tau + \frac{\alpha_{p+1}^\tau X_{p+1}^\tau - \alpha_p^\tau X_p^\tau}{\alpha_{p+1}^\tau + \alpha_p^\tau}, \\ X_{p+1}^{\tau+1} &= X_{p+1}^\tau - \frac{\alpha_{p+1}^\tau X_{p+1}^\tau - \alpha_p^\tau X_p^\tau}{\alpha_{p+1}^\tau + \alpha_p^\tau}. \end{aligned} \tag{5}$$

On substituting $p=1$, it can be seen after some elementary calculations that this equals Eq. (3).

We will present two local strategies, the generalized dimension exchange method (GDE) and a diffusive method. The latter will be given first for the discrete case of a finite number of PE's. Then we give a description for the continuous case that the number of processors tends to infinity. This will provide a motivation for the multi-level strategy to be presented in Section 5.4. In Section 5.5 we give a global strategy. In the last subsection we spend some words on partial balancing and grid-refinement.

Remark 5.1. In the rest of this work we assume that all domain-distributions obtained in the process of domain-balancing can actually be allocated on the PE under consideration. In practice, this may lead to complications, since it may happen that not enough memory is available on PE p to store $X_p^{\tau+1}$.

5.2. *The generalized dimension exchange method*

In the generalized dimension exchange method (GDE) (introduced by Xu and Lau [7] for the case of homogeneous dedicated clusters) the processor topology is considered a graph with vertices the PE's and edges connecting those PE's which

have to exchange information during the computation (in our case: which have to exchange information concerning the update of the dummy-points). These edges are colored in such a way that no two equally colored edges leave one vertex. The number of colors is called the dimension of the graph.

Domain-balancing is obtained by — successively for each color — exchange of columns between processors connected by an edge of that particular color. In our case of a linear PE-array, we need only two colors. The domain-balancing alternately involves exchange of columns between PE's belonging to the pairs

$$2p+1 \text{ and } 2p+2, \quad p=0, \ldots, \lfloor \frac{P}{2} \rfloor, \text{ (the odd phase)}$$

and

$$2p+2 \text{ and } 2p+3, \quad p=0, \ldots, \lfloor \frac{P}{2} \rfloor - 1, \text{ (the even phase).}$$

Here $\lfloor x \rfloor$ denotes the largest integer smaller than x. Because in each phase of the balancing only pairs of two neighboring PE's balance their domain, this can be done using Eq. (5). It is easily verified that this reduces to the GDE method of Xu and Lau [7] for the case of a linear PE-array. (Indeed, reformulation of Eq. (5) in $X_p^{\tau+1} = (1-\lambda)X_p^{\tau} + \lambda X_{p+1}^{\tau}$ (the form as used in Ref. [7]) shows that $\lambda = \alpha_p^{\tau}/(\alpha_{p+1}^{\tau} + \alpha_p^{\tau})$ depends on p as it should for a heterogeneous workstation-cluster.) A moment of reflection shows that this method can be written as

$$\boldsymbol{X}^{\tau+1} = \boldsymbol{\alpha}_1^{\tau}\boldsymbol{\alpha}_2^{\tau}\boldsymbol{X}^{\tau}, \tag{6}$$

where $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ correspond to the two colors. Both are $P \times P$ block-diagonal matrices, with block-sizes 1 and 2, depending on whether P is even or odd. The (tridiagonal) matrix $\boldsymbol{\alpha}_1\boldsymbol{\alpha}_2$ has the following properties:

(i) The column-sums are all equal to 1, since this holds for $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$.

(ii) The entries are all strictly positive.

(iii) It is irreducible.

Standard Perron–Frobenius theory reveals that in the static case (i.e. $\boldsymbol{\alpha}_1$, $\boldsymbol{\alpha}_2$ not dependent on τ), the iteration Eq. (6) converges to the eigenvector corresponding to eigenvalue 1. Direct inspection shows that the components of this eigenvector correspond to the optimal distribution for a dedicated cluster, as given by Eq. (3). Note that several generalizations are possible, e.g. the k-step GDE method, where $\boldsymbol{X}^{\tau+1} = (\boldsymbol{\alpha}_1\boldsymbol{\alpha}_2)^k\boldsymbol{X}^{\tau}$.

5.3. *Diffusion method*

5.3.1. *The discrete case*

Motivated by Eq. (5), in the general case we put for $1 < p < P$,

$$X_p^{\tau+1} = X_p^{\tau} + \beta\left(\frac{\alpha_{p+1}^{\tau}X_{p+1}^{\tau} - \alpha_p^{\tau}X_p^{\tau}}{\alpha_{p+1}^{\tau} + \alpha_p^{\tau}}\right) + \beta\left(\frac{\alpha_{p-1}^{\tau}X_{p-1}^{\tau} - \alpha_p^{\tau}X_p^{\tau}}{\alpha_{p-1}^{\tau} + \alpha_p^{\tau}}\right), \tag{7}$$

and for $p=1$ resp. $p=P$,

$$X_1^{\tau+1} = X_1^{\tau} + \beta\left(\frac{\alpha_2^{\tau} X_2^{\tau} - \alpha_1^{\tau} X_1^{\tau}}{\alpha_2^{\tau} + \alpha_1^{\tau}}\right), \tag{8}$$

$$X_P^{\tau+1} = X_P^{\tau} + \beta\left(\frac{\alpha_{P-1}^{\tau} X_{P-1}^{\tau} - \alpha_P^{\tau} X_P^{\tau}}{\alpha_P^{\tau} + \alpha_{P-1}^{\tau}}\right). \tag{9}$$

Here β is some parameter, which is specified later. Eqs. (7)–(9) can be cast into the following shorthand form:

$$\boldsymbol{X}^{\tau+1} = \boldsymbol{\alpha}^{\tau}(\beta)\boldsymbol{X}^{\tau}, \tag{10}$$

where $\boldsymbol{\alpha}^{\tau}(\beta)$ is a $P \times P$ tridiagonal matrix. More generally, we can introduce the k-step diffusion method as

$$\boldsymbol{X}^{\tau+1} = (\boldsymbol{\alpha}^{\tau}(\beta))^k \boldsymbol{X}^{\tau}$$

However, for a convergence analysis it suffices to consider the case $k=1$. The matrix $\boldsymbol{\alpha}^{\tau}(\beta)$ satisfies the following properties:

(i) the column-sums are all equal to 1,

(ii) if $\beta < \frac{1}{2}$, all entries are positive,

(iii) it is irreducible, since all $\alpha_p^{\tau} > 0$.

Again, by standard Perron–Frobenius theory, the eigenvalues of $\boldsymbol{\alpha}$ lie within the unit circle; the only eigenvalue on the unit circle is 1. If $\boldsymbol{\alpha}$ does not depend on τ (stationary load due to the other tasks) the iteration Eq. (10) converges towards the eigenvector corresponding to eigenvalue 1. Direct verification yields that this eigenvector is precisely the static optimal distribution as given by Eq. (3). The convergence speed is determined by the second largest eigenvalue. A straightforward analysis of the case $P=3$ reveals that $\beta=\frac{1}{2}$ is optimal, if we take into account that the loads X_p^{τ} are positive, leading to the requirement $\beta \leq \frac{1}{2}$. So in the sequel we take $\beta=\frac{1}{2}$ and denote $\boldsymbol{\alpha}^{\tau}(\frac{1}{2})$ by $\boldsymbol{\alpha}^{\tau}$. Concerning this second largest eigenvalue, Boillat [14] has shown for a homogeneous cluster and a slightly different strategy that this eigenvalue converges essentially as $(1-1/P^2)$ for $P \to \infty$, which means that the convergence towards the optimal distribution becomes very slow. For our strategy a similar result will hold (no proof).

For that reason, Horton [8] proposed a multi-level strategy. We will present a modified version of this in the next subsection. First, for the sake of motivation, and because it is a nice result in itself, we will give a description of the behavior of the local strategy for large P.

Remark 5.2. The GDE method and the diffusion method can be implemented in both a synchronous and an asynchronous way. In the synchronous way, all PE's do the balancing simultaneously. In the asynchronous implementation, each PE does a balancing at the moment that it reaches a balancing statement and finds (by exchanging timings with its neighbors) that a balancing has to take place.

5.3.2. The continuous case

Put $h=1/P$, and let X_h, α_h:$(t, \xi) \in \mathbb{R}^+ \times [0, 1] \to \mathbb{R}$ be interpolating functions such that

$$X_h\left(\tau \frac{h^2}{4}, ph\right) = X_p^\tau, \quad p=1, \ldots, P, \ \tau=0, 1, 2, \ldots,$$

and

$$\alpha_h\left(\tau \frac{h^2}{4}, ph\right) = \alpha_p^\tau, \quad p=1, \ldots, P, \ \tau=0, 1, 2, \ldots.$$

Then,

$$\alpha_{p+1}^\tau X_{p+1}^\tau - \alpha_p^\tau X_p^\tau \approx h\partial_\xi(X_h\alpha_h)\left(\tau \frac{h^2}{4}, \left(p+\frac{1}{2}\right)h\right),$$

where $\partial_\xi f$ denotes differentiation with respect to ξ of f. Further,

$$\alpha_{p+1}^\tau + \alpha_p^\tau \approx 2\alpha_h\left(\tau \frac{h^2}{4}, \left(p+\frac{1}{2}\right)h\right).$$

So, after some manipulations we obtain from Eq. (7) (with $\beta=1/2$),

$$X_p^{\tau+1} = X_h\left(\tau \frac{h^2}{4} + \frac{h^2}{4}, ph\right) \approx X_h\left(\tau \frac{h^2}{4}, ph\right) + \frac{h^2}{4}\partial_\xi\left(\frac{\partial_\xi(X_h\alpha_h)}{\alpha_h}\right)\left(\tau \frac{h^2}{4}, ph\right),$$

which happens to be the Euler discretization with step-length $h^2/4$ of the Fokker–Planck equation

$$\partial_t X = \partial_\xi\left(\partial_\xi X + X\frac{\partial_\xi \alpha}{\alpha}\right) = \partial_\xi\left(\frac{1}{\alpha}\partial_\xi(\alpha X)\right), \tag{11}$$

where we have put

$$X = \lim_{h\to 0} X_h, \quad \text{and} \quad \alpha = \lim_{h\to 0} \alpha_h.$$

Note that in this derivation we assumed that these limits exist, which need not be the case for arbitrary sequences X_h, α_h.

In order to conclude the continuous formulation, we have to supply Eq. (11) with boundary conditions for $X(\cdot, 0)$ and $X(\cdot, 1)$. A natural boundary condition can be obtained from the discrete case for $p=1$ resp. $p=P$. From Eq. (8) it follows that e.g.

$$X_h\left(\tau \frac{h^2}{4} + \frac{h^2}{4}, 0\right) \approx X_h\left(\tau \frac{h^2}{4}, 0\right) + \frac{h^2}{4}\frac{\partial_\xi(X_h\alpha_h)(\tau(h^2/4), 0)}{h}.$$

This is consistent with Eq. (11) if this is an Euler discretization with step-size $h^2/4$.

But then

$$\lim_{h\to 0} \frac{\partial_\xi(X_h\alpha_h)(\tau(h^2/4),\,0)}{h}$$

must exist, which is the case if

$$\partial_\xi(X\alpha)(\cdot\,,\,0)=0, \tag{12}$$

and similarly

$$\partial_\xi(X\alpha)(\cdot\,,\,1)=0. \tag{13}$$

We check that the total load is conserved, i.e.

$$\partial_t\int_0^1 X(\cdot\,,\,\xi)\,\mathrm{d}\xi=0.$$

This follows from Eq. (11), since

$$\partial_t\int_0^1 X(\cdot\,,\,\xi)\mathrm{d}\xi=\left(\partial_\xi X+X\frac{\partial\alpha}{\alpha}\right)\Bigg|_0^1=\frac{\partial_\xi(\alpha X)}{\alpha}\Bigg|_0^1=0,$$

by the boundary conditions Eqs. (12) and (13).

Let us consider some examples.

Example 1. First we consider the stationary state of Eq. (11). Then $\partial_t X=0$, and from Eq. (11) and the boundary condition Eq. (12) it follows that $X\alpha=k(t)$ for some function $k(t)$, which represents the optimal situation where each PE finishes its part of our job in an equal amount of time. Note that the case of time-dependent $\alpha(t,\,\xi)$ is allowed. This corresponds e.g. to a situation where another balanced job starts or finishes on the cluster during the run of our job.

Example 2. For a dedicated homogeneous cluster, Eq. (11) reduces to the diffusion equation,

$$\partial_t X=\partial_\xi^2 X,\ \partial_\xi X(\cdot\,,\,0)=\partial_\xi X(\cdot\,,\,1)=0.$$

The general solution of this equation can be obtained by separation of variables. Putting $X(t,\,\xi)=T(t)\Xi(\xi)$, we obtain a Sturm–Liouville problem for Ξ. This provides us with the existence of the characteristic times $\lambda_k=k\pi$, $k=0,\,1,\,2,\,\ldots$. The general solution is a superposition

$$\sum_{k=0}^{\infty} B_k\mathrm{e}^{-\lambda_k t}\cos(\lambda_k\xi),$$

where the constants B_k are determined from the initial condition at $t=0$.

For a heterogeneous cluster and α not dependent on t, Eq. (11) with boundary conditions Eqs. (12) and (13) can also be solved by separating variables, resulting again in a Sturm–Liouville problem for Ξ. So in that case as well there exist characteristic times λ_k, $k=0,\,1,\,2,\,\ldots$ and characteristic functions Ξ_k such that the

solution can be written as

$$\sum_{k=0}^{\infty} B_k e^{-\lambda_k t} \Xi_k(\xi).$$

Remark 5.3. Example 2 seems to be in contradiction with the result obtained by Boillat concerning the convergence of the discrete case. ¿From example 2 it follows that the convergence of the continuous case is not worse than $e^{-\pi t}$, whereas in the discrete case the convergence-speed tends to zero if $P \to \infty$. The reason is that the discrete case in fact is a discretization of the continuous case. For such discretizations it is well-known that the low-frequency components of the discretization error are only very slowly damped. In fact, this is precisely the motivation for the development of multigrid methods for the numerical solution of PDE's (cf. Ref. [15]). In these methods, the low-frequency components of the error are reduced using coarser grids. The application of multigrid methods to dynamic load-balancing results in the multilevel strategy of the following section.

5.4. *Multilevel strategy*

The multilevel method is based on the following reasoning. Suppose we have an unbalanced work-distribution. Partition the set $\mathscr{P} = \{1, \ldots, P\}$ into two subsets $\mathscr{P}_1$ and $\mathscr{P}_2$. In general we can assume that

$$\max_{p \in \mathscr{P}_1} T_p > \max_{p \in \mathscr{P}_2} T_p.$$

We now transfer some columns from $\mathscr{P}_1$ to $\mathscr{P}_2$, which are redistributed over the PE's in $\mathscr{P}_1$ and $\mathscr{P}_2$ in such a way that each of the PE's decreases or increases its load X_p linearly. More precisely, if x_1 is the new number of columns in $\mathscr{P}_1$ and the new domain-distribution is denoted by $\hat{X}$, then for all $p \in \mathscr{P}_1$:

$$\hat{X}_p = \frac{x_1}{\sum_{p \in \mathscr{P}_1} X_p} X_p,$$

and we define

$$\hat{T}_p = \alpha_p \hat{X}_p.$$

The number of columns to be transferred is determined by the requirement that

$$\max_{p \in \mathscr{P}_1} \hat{T}_p = \max_{p \in \mathscr{P}_2} \hat{T}_p.$$

Now we apply the same procedure to $\mathscr{P}_1$ and $\mathscr{P}_2$, with X_p replaced by $\hat{X}_p$, for $p = 1, \ldots, P$. This results in a globally balanced state after a finite number of steps, as is proved in Theorem 5.1 below.

We need one more notation: if $\boldsymbol{X} = (X_{k_1}, \ldots, X_{k_2})$, then the length of $\boldsymbol{X}$ is denoted

by $\|\boldsymbol{X}\|$:

$$\|\boldsymbol{X}\| = \sum_{i=k_1}^{k_2} X_i.$$

Now we give the multilevel balancing method as a recursive procedure, and prove its convergence.

procedure balance(p_f, p_l, $\boldsymbol{X}^{\text{old}}$, $\boldsymbol{X}^{\text{new}}$, $\boldsymbol{\alpha}$)
if $p_f = p_l$ **then** $X^{\text{new}}_{p_f} = X^{\text{old}}_{p_f}$
else
Partition $\mathscr{P} = \{p_f, \ldots, p_l\}$ into $\mathscr{P}_1$ and $\mathscr{P}_2$, where
$\mathscr{P}_1 = \{p_f, \ldots, p_m\}$, $\mathscr{P}_2 = \{p_m+1, \ldots, p_l\}$, with $p_m = \lceil (p_l - p_f)/2 \rceil$.
Determine numbers x_1 and x_2 such that
$x_1/\|\boldsymbol{X}_1\| \max_{p\in\mathscr{P}_1} \alpha_p X_p = x_2/\|\boldsymbol{X}_2\| \max_{p\in\mathscr{P}_2} \alpha_p X_p$
$x_1 + x_2 = \|\boldsymbol{X}_1\| + \|\boldsymbol{X}_2\|$.
for $p \in \mathscr{P}_1$ put $\hat{X}_p = (x_1/\|\boldsymbol{X}_1\|) X_p$
for $p \in \mathscr{P}_2$ put $\hat{X}_p = (x_2/\|\boldsymbol{X}_2\|) X_p$
balance(p_f, p_m, $\hat{\boldsymbol{X}}$, $\boldsymbol{X}^{\text{new}}$, $\boldsymbol{\alpha}$)
balance(p_m+1, p_l, $\hat{\boldsymbol{X}}$, $\boldsymbol{X}^{\text{new}}$, $\boldsymbol{\alpha}$)
endif
end

Theorem 5.1. Let $\mathbf{X}_0$ be arbitrary, and let $\mathbf{X}_l$, $l = 1, 2, \ldots$, be given by

balance(1, P, $\boldsymbol{X}_{l-1}$, $\boldsymbol{X}_l$, $\boldsymbol{\alpha}$).

Then for $l = \lceil \log_2(P) \rceil$ the domain distribution $\boldsymbol{X}_l$ equals the globally balanced state $\bar{\boldsymbol{X}}$, which satisfies

$$\alpha_p \bar{X}_p = \alpha_{p+1} \bar{X}_{p+1}, \quad p = 1, \ldots, P-1$$

$$\|\bar{X}\| = \|X_0\|.$$

Proof. The proof rests on two observations. First notice that if in the partition step, $\mathscr{P}_1$ and $\mathscr{P}_2$ happen to be such that

$$\alpha_p X_p = \alpha_{p+1} X_{p+1}, \quad p = p_f, \ \ldots, p_m - 1$$

$$\alpha_p X_p = \alpha_{p+1} X_{p+1}, \quad p = p_m, \ldots, p_l - 1,$$

then also — since $\hat{X}_p$ is a multiple (independent of p) of X_p —

$$\alpha_p \hat{X}_p = \alpha_{p+1} \hat{X}_{p+1}, \quad p = p_f, \ldots, p_m - 1 \quad \text{and} \quad \alpha_p \hat{X}_p = \alpha_{p+1} \hat{X}_{p+1}, \quad p = p_m, \ldots, p_l - 1,$$

and so $X^{\text{new}}_p = \hat{X}_p$, $p = p_f, \ldots, p_l$, and by construction of x_1 and x_2,

$$\alpha_p X^{\text{new}}_p = \alpha_{p+1} X^{\text{new}}_{p+1}, \quad p = p_f, \ \ldots, \ p_l - 1.$$

Second, after one call balance(1, P, $\boldsymbol{X}_0$, $\boldsymbol{X}_1$, $\boldsymbol{\alpha}$), the subsets consisting of 1 or 2 consecutive PE's are balanced. By induction it follows that after $\lceil \log_2(P) \rceil$ calls the globally balanced state is obtained.

The actual domain-balancing is done by

$X_0 = X^\tau$
do $i = 1, l$ balance$(1, P, X_{l-1}, X_l, \alpha^\tau)$ **enddo**
$X^{\tau+1} = X_l$.

This strategy will be referred to as the l-sweep multilevel strategy.

Remark 5.4. This multilevel method is different from the one proposed by Horton [8]. In fact, the latter method amounts to determination of the ideally balanced state $\bar{X}$ satisfying

$$\alpha_p^\tau \bar{X}_p^\tau = \alpha_{p+1}^\tau \bar{X}_{p+1}^\tau, \quad p = 1, \ldots, P-1.$$

However, we can determine $\bar{X}^\tau$ explicitly, by using Eq. (3). This is the basis of the global method presented in the next subsection.

5.5. *The global strategy*

Besides the iterative methods mentioned so far, there is a direct method to obtain the globally balanced state.

Let $\bar{X}^\tau$ be the solution of

$$\alpha_p^\tau \bar{X}_p^\tau = \alpha_{p+1}^\tau \bar{X}_{p+1}^\tau, \quad p = 1, \ldots, \quad P-1$$
$$\|\bar{X}^\tau\| = \|X^\tau\|.$$

The solution $\bar{X}^\tau$ is given by Eq. (3), the computation of which requires a global sum, showing the global character of the method. This method will be called the *exact global method.*

5.6. *Full and partial balancing, grid refinement*

All methods mentioned above have the following general form:

$$X^{\tau+1} = F_k(X^\tau), \tag{14}$$

where the symbol F_k refers to e.g. the k-step diffusion method, etc. (For the exact global method, $k = 1$.) For all these methods

$$\lim_{k\to\infty} F_k(X^\tau) = \bar{X}^\tau,$$

and hence they all reduce to the exact global method if $k \to \infty$.

Under some circumstances, it is better to do a partial balancing, because all balancing methods rely on an estimate for $\alpha^{\tau+1}$. If α^τ is rapidly fluctuating in time, these estimates are likely to be inaccurate. Therefore we introduce the one-parameter families of balancing methods given by

$$X^{\tau+1} = (1-\lambda)X^\tau + \lambda F_k(X^\tau), \tag{15}$$

where $0 \le \lambda \le 1$, and denote them by the terms *partial balancing k-step diffusion method*, etc. Note that if $\lambda = 0$, no balancing is done. If $\lambda = 1$ we speak of full balancing.

We finally spend some words on grid refinement. Due to the general form

(Eq. (14)), a grid refinement consisting of adding or deleting some extra columns of grid-points to computational space can be easily incorporated. If e.g. ζ_p^τ is the number of extra columns inserted in $\tilde{A}_p$ on processor p at time-level τ, we obtain

$$X^{\tau+1} = F_k(X^\tau + \zeta^\tau) \quad \text{and} \quad N_1^{\tau+1} = N_1^\tau + \sum_{p=1}^{P} \zeta_p^\tau.$$

Note that adding or deleting rows leads to a change in N_2, and not to a change in the column-distribution X.

6. Performance of the load-balancing methods

In this section we consider the effectivity of the domain-balancing methods mentioned in the previous section.

6.1. Introduction

In the application of domain-balancing, for all methods the following phases can be distinguished:

(i) *Computation step*: perform a Runge–Kutta stage for time-level τ with the given domain-distribution X^τ.

(ii) *Statistics step*: measure T_p^τ, $p=1, \ldots, P$, and determine α_p^τ, $p=1, \ldots, P$.

(iii) *Decision step*: decide whether or not to redistribute the domain and if appropriate, which method to use.

(iv) *Balancing step*: Calculate the new domain-distribution $X^{\tau+1}$ by application of the chosen method.

(v) *Redistribution step*: Redistribute the domain over the PE's.

In a practical situation, the execution of our job can be delayed on the one hand in the computation step due to a non-optimal domain distribution, and on the other hand due to the communication and synchronization costs required in the redistribution step. Therefore, the most important step is the decision step, where this trade-off must be analyzed. Suppose at time-level τ we have a domain-distribution X^τ, and a load given by α^τ. By domain-balancing, we obtain a distribution $X^{\tau+1}$, which, in general, will not be balanced with respect to $\alpha^{\tau+1}$. Due to the fact that the next time-level is calculated with the unbalanced domain, we obtain a throughput-delay of

$$C_{\text{comp}}^\tau = \left(\max_{p=1,\ldots,P} \alpha_p^{\tau+1} X_p^{\tau+1} \right) - \alpha_p^{\tau+1} \bar{X}_p^{\tau+1},$$

where $X^{\tau+1}$ is the balanced domain w.r.t. $\alpha^{\tau+1}$. However, also the balancing costs time, which we will denote by C_{bal}^τ. The total delay arising at time-level τ is then

$$\Delta^\tau = C_{\text{comp}}^\tau + C_{\text{bal}}^\tau. \tag{16}$$

In fact, a domain-balancing method must minimize

$$\sum_{\tau} \Delta^{\tau}. \tag{17}$$

In the rest of this section, we will first analyze Eq. (16) somewhat further, and focus on the effect of the CFD problem-size, the actual implementation of the algorithm, and the characteristics of the workstation-cluster on the choice of a domain-balancing method. Next we will discuss the influence of the statistical nature of α^{τ} on the performance of the domain-balancing strategies. We introduce the balancing efficiency and the balancing speedup in order to quantify this performance. Finally, we will give an example illustrating these points.

6.2. *Analysis of the delay at one time-level*

In this subsection we analyze Eq. (16), in order to obtain an impression of the influence of the problem-parameters (f, N_1, N_2, N_3), the hardware-characteristics (B, S) and the load-variations (l_p^{τ}) on the delay Δ^{τ} at time-level τ. First we derive an algorithm for the computation of C_{bal}^{τ}.

Theorem 6.1. Let $\mathbf{X}=X_1, \ldots, X_P$, and $\mathbf{Y}=Y_1, \ldots, Y_P$, with $\|\mathbf{X}\|=\|\mathbf{Y}\|$. The number of columns to be transferred to redistribute $\mathbf{X}$ into $\mathbf{Y}$ is at most

$$d(\mathbf{X}, \mathbf{Y})=\sum_{p=1}^{P}\left|\sum_{i=1}^{p} X_i - Y_i\right|.$$

Proof. In order to obtain Y_1 from X_1 we have to transfer $C=|X_1-Y_1|$ columns from PE 1 to PE 2. After this transfer, we have on PE 2 some intermediate number of columns $\tilde{X}_2=X_2+X_1-Y_1$. So the number of columns to be transferred to PE 3 is $|\tilde{X}_2-Y_2|$. On PE 3 we then obtain the intermediate number of columns $\tilde{X}_3=X_3+\tilde{X}_2-Y_2$. By induction, the total number C of columns to be transferred is obtained by the following algorithm:

$C=|X_1-Y_1|$
$\tilde{X}_1=X_1$
do $p=2, P-1$
  $\tilde{X}_p=X_p+\tilde{X}_{p-1}-Y_{p-1}$
  $C=C+|Y_p-\tilde{X}_p|$
end do

In the p-th step we get

$$C=C+|Y_p-X_p+Y_{p-1}-\tilde{X}_{p-1}|=C+|Y_p-X_p+Y_{p-1}-X_{p-1}+Y_{p-2}-X_{p-2}+\ldots+Y_1-X_1|,$$

proving the theorem.

For the case of Navier–Stokes equations we obtain

$$C_{\text{bal}}^{\tau}(NS)=\frac{N_2 N_3}{B}\, d(\mathbf{X}^{\tau}, \mathbf{X}^{\tau+1}).$$

Now we consider the term C^{τ}_{comp} in Eq. (16), again for the Navier–Stokes equations (the Burgers case is treated similarly). Since

$$\alpha_p^{\tau+1} = \frac{N_2 N_3 f}{S_p}(1 + l_p^{\tau+1}),$$

we obtain

$$C^{\tau}_{\text{comp}}(NS) = N_2 N_3 f \max_{p=1,\ldots,P} \frac{1 + l_p^{\tau+1}}{S_p}(X_p^{\tau+1} - \bar{X}_p^{\tau+1}).$$

For a homogeneous cluster, Eq. (16) becomes

$$\Delta^{\tau} = \frac{N_2 N_3}{B}\left(\frac{fB}{S} \max_{p=1,\ldots,P}(1 + l_p^{\tau+1})(X_p^{\tau+1} - \bar{X}_p^{\tau+1}) + d(\boldsymbol{X}^{\tau}, \boldsymbol{X}^{\tau+1})\right), \tag{18}$$

which will be written as

$$\Delta^{\tau} = \gamma\left(\frac{fB}{S}\hat{C}^{\tau}_{\text{comp}} + \hat{C}^{\tau}_{\text{bal}}\right).$$

We can draw the following conclusions related to the decision which balancing method must be used:

(i) (Concerning the problem-size.) The problem-size is not important. Note that this is due to the linearity of the balancing methods (ensuring independence of N_1) and to the fact that both the transfer of a slab of the computational space and the computation of $u_{ijk}^{\tau+1}$ out of u_{ijk}^{τ} for all j, k are proportional to $N_2 N_3$.

(ii) (Concerning the computational intensity f.) If the number f increases, it is increasingly better to do a full domain-balancing aiming at $\boldsymbol{X}^{\tau+1} \approx \bar{\boldsymbol{X}}^{\tau+1}$. If the number f is rather low, and the load-fluctuations are high, partial balancing will be preferable.

(iii) (Concerning the hardware-characteristics.) If the ratio of the bandwidth to computation speed decreases, domain-balancing might not be a good idea, since for low enough B/S the balancing costs become dominant in Eq. (18). However, partial balancing might still diminish Δ^{τ}.

6.3. *Performance of the balancing methods*

In order to quantify the performance of the various balancing methods, we introduce the balancing speedup σ. Given α_p^{τ} for some time-level τ, the ideal completion time for Runge–Kutta stage τ is

$$\frac{\sum_{p=1}^{P} X_p^{\tau}}{\sum_{p=1}^{P} 1/\alpha_p^{\tau}} = \frac{N_1}{\sum_{p=1}^{P} 1/\alpha_p^{\tau}}.$$

So if we have a run of, say, k time-levels, the ideal completion time $T_{\text{ideal}}(k)$ is

$$T_{\text{ideal}}(k) = \sum_{\tau=1}^{k} \frac{N_1}{\sum_{p=1}^{P} 1/\alpha_p^{\tau}}.$$

However, due to the time spent in the domain-balancing, the real completion time $T_{\text{real}}(k)$ will be larger. If balancing is successful, T_{real} should be smaller than the time $T_{\text{no-lb}}$ used for completion of k time-levels without balancing. The *balancing speedup* $\sigma(k)$ is defined as

$$\sigma(k) = \frac{T_{\text{no-lb}}(k)}{T_{\text{real}}(k)}.$$

6.4. Simulation of the balancing methods

Suppose we want to carry out a CFD-simulation on a given cluster, and assume that we have some information about the nature of the load-fluctuations on the cluster. Given the information of the previous subsections, some insight might already be present about which balancing methods can be used. For example, if the load-fluctuations are very rapid, partial balancing is probably to be preferred above full balancing. Or, if the load fluctuates only slowly, and/or the number f is very large, full balancing can be considered. However, even if the nature of the load-fluctuations is known quite well, it is very likely that a priori determination of an optimal balancing method is hardly possible. In a practical situation, it is of course not possible to run a CFD-job with each of the balancing methods, since the time needed to run such a job can be rather large. Therefore, we propose to simulate the combination of the CFD-computation and the balancing methods, to find the optimal one.

We will illustrate this by an example of a cluster with a very simple fluctuating load. We will consider both the solution of Burgers' equation and the integration of the 3d Navier–Stokes equations, as considered in Section 4.2. In the first case, f is rather small, so Δ^{τ} is most likely to be dominated by the balancing term, whereas in the latter case f is large, so the computation term will be most important.

We use the following simplifying assumptions:

(i) The cluster is homogeneous.
(ii) The communication of the dummy-points is hidden by the computation (i.e. $P < P^*$ in the notation of Section 4.1).
(iii) The statistics, decision and balancing-step cost a negligible amount of time.
(iv) The redistribution step cannot be hidden by the computation step.
(v) The other jobs do not cause paging or saturation of the network.

Recall that l_p is the number of other jobs running on PE p. For simplicity we will only consider synchronous implementations of the local balancing strategies. Then the time to complete one Runge–Kutta stage and to perform the load-balancing (if

applied) is

$$\max_{p=1,\ldots,P} \alpha_p X_p + T_{lb},$$

where T_{lb} is the time for the redistribution, and α_p is given by

$$\alpha_p = \frac{(1+l_p)N_2 f}{S_p},$$

in the Burgers case, and by

$$\alpha_p = \frac{(1+l_p)N_2 N_3 f}{S_p},$$

in the Navier–Stokes case.

The balancing time T_{lb} is assumed to be the total number of bytes transferred over the network divided by the bandwidth. We will calculate the throughput-time for 1000 Runge–Kutta stages on a homogeneous cluster with $P=6$ workstations, each with a sustained speed of $S_p=10$ Mflops, connected on a network with a sustained bandwidth of $B=0.15$ Mword per second. In the case of Burgers' equation, we choose an algorithm requiring $f=40$ floating point operations in order to calculate $u_{ij}^{\tau+1}$ out of u_{ij}^{τ}. Further $N_2=300$, and we take $N_1=300$. In the Navier–Stokes case, we choose $f=500$, $N_1=N_2=N_3=128$. In both cases $P<P^*$.

In the example each PE of the cluster is used by one other user for half of the number of time-levels. The load-functions l_p^{τ} differ only in the length of the time-intervals at which the other user is active, and are given by

$$l_p^{\tau} = \begin{cases} 0, & \tau \bmod \lceil 200/p \rceil < \lceil 100/p \rceil \\ 1, & \text{otherwise} \end{cases} \tag{19}$$

With this load, $T_{\text{ideal}}=90$ s for the Burgers case, and $T_{\text{ideal}}=26214$ s for the Navier–Stokes case.

The performance of the various domain-balancing strategies is summarized in Fig. 3 for the Burgers case (left picture), and for the Navier–Stokes case (right picture).

It can be seen that in the case of a small f domain-balancing will not give considerable speed-up. On the contrary: application of domain balancing may even slow down the execution of the CFD-job. In the case of Navier–Stokes equations, where we need a rather high number f, domain-balancing may speed up the calculations in this case by some 25% (which, for this run, implies a saving in wall-clock time of about 2 hours).

Further it can be deduced that for both cases the family of global methods can provide best results, provided the optimal value for the parameter λ is chosen. From the simulations we conclude for the Navier–Stokes equations that, given the load on this cluster, domain-balancing is advisable; the recommended method in this case would be the exact global method.

For other load-situations, a simulation like the one we performed will result in a

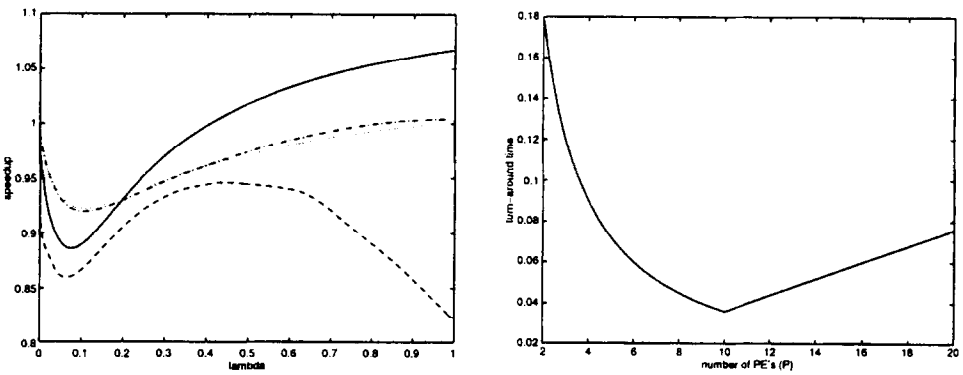

Fig. 3. Speed-up of the various 1-step domain-balancing strategies for the case of Burgers' equation (left picture) and Navier–Stokes equations (right picture). Shown are the speed-ups for the family of partially balancing methods. Lambda (λ) is the parameter in Eq. (15) which denotes the amount of balancing. The line-styles refer to the global method (solid), the diffusion method (dotted), the GDE method (dash–dotted) and the multilevel method (dashed).

recommendation on whether to employ domain-balancing, and if so, which method should be used. Concerning all methods, the results presented here suggest that there is an optimal λ, depending on the nature of the load-fluctuations. One might get the impression that this optimum is $\lambda = 1$ in the case of the global method, but simulations for other load-situations revealed that this is not generally true. Estimation of this optimal λ will be the subject of future research.

In the limit for $k \to \infty$, all methods should perform equally well. In Fig. 4 it is shown for the diffusion method that if $k \to \infty$, the behavior of the family of partially balanced k-step methods approaches that of the family of global methods.

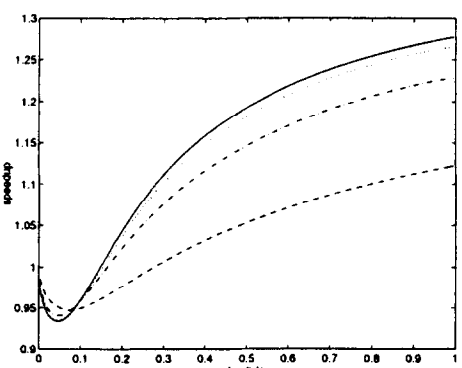

Fig. 4. Speed-up for the family of partially balanced k-step diffusion methods applied to the Navier–Stokes equations for $k = 1$ (dashed), $k = 6$ (dash–dotted), $k = 16$ (thin dots) and $k = \infty$ (and global method) (solid).

7. Conclusions

We have considered the application of domain-decomposition techniques for flow-simulation of workstation clusters. We studied dynamic adaptation of the sizes of the domains, depending on the activities on the cluster. We have discussed several dynamic load-balancing algorithms known from literature and adapted them for this case. Moreover, we introduced a new one-parameter family of load-balancing methods. According to a model study of the performance none of these algorithms is the best under *arbitrary* load-circumstances. If the amount of computational work in the flow-simulation per time-step is small, then dynamic load-balancing will not give reasonable speedup, whereas it may give some speedup in the case of much computational work per time-step. Therefore, dynamic load-balancing has to be applied with some care.

References

[1] H.M. Willebeek-Le Mair, A.P. Reeves, IEEE Trans. Parallel Distr. Syst. 4 (1993) 979–993.
[2] J. Schabernack, Informationstechnik (IT) 34 (1992) 280–295.
[3] A.Y. Grama, V. Kumar, N.R. Vempaty, J. Parallel Distr. Comput. 22 (1994) 60–79.
[4] R.D. Williams, Concurrency: Practice Experience 3 (1991) 457–481.
[5] C.Z. Xu, F.C.M. Lau, J. Operational Res. Soc. 45 (1994) 786–796.
[6] G. Cybenko, J. Parallel Distr. Comput. 7 (1989) 279–301.
[7] C.Z. Xu, F.C.M. Lau, J. Parallel Distr. Comput. 16 (1992) 385–393.
[8] G. Horton, Parallel Comput. 19 (1993) 208–218.
[9] B.J. Geurts, H. Kuerten, J. Eng. Math. 27 (1993) 293–307.
[10] R.D. Ferraro, P.C. Liewer, V.K. Decyk, J. Comput. Phys. 109 (1993) 329–341.
[11] C.H. Cap, V. Strumpen, Parallel Comput. 19 (1993) 1221–1234.
[12] N. Nedeljkovic, M.J. Quinn, Concurrency: Practice Experience 5 (1993) 257–268.
[13] F.C.H. Lin, R.M. Keller, IEEE Trans. Software Eng. 13 (1987) 32–38.
[14] J.E. Boillat, Concurrency: Practice Experience 2 (1990) 289–313.
[15] P. Wesseling, An Introduction to Multigrid Methods, John Wiley, 1992.