SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 9, Number 1, pp. 61–68. http://www.scpe.org
ISSN 1895-1767

LOAD BALANCING FOR THE NUMERICAL SOLUTION OF THE NAVIER-STOKES EQUATIONS

GREGORY KARAGIORGOS* AND NIKOLAOS M. MISSIRLIS*

Abstract. In this paper we simulate the performance of a load balancing scheme. In particular, we study the application of the Extrapolated Diffusion (EDF) method for the efficient parallelization of a simple 'atmospheric' model. Our model involves the numerical solution of the steady state Navier-Stokes (NS) equations in the horizontal plane and random load values, corresponding to the "physics" computations, in the vertical plane. For the numerical solution of NS equations we use the Local Modified Successive Overrelaxation (LMSOR) method with local parameters thus avoiding the additional cost caused by the global communication of the involved parameter ω in the classical SOR method. We have implemented an efficient domain decomposition technique by using a larger number of processors in the areas of the domain with heavier work load. Our results show that in certain cases we have a gain as much as approximately 45% in execution time when our load balancing scheme is applied.

Key words. load balancing, domain decomposition, diffusion method.

1. Introduction. The efficient use of the available computational resources for running parallel applications in a distributed computing environment leads to the load balancing problem. Many parallel applications produce different workload during execution time. Parallel weather prediction simulations constitute typical applications that require load balancing techniques. However, the load has to be balanced in order to achieve an efficient use of the processor network. In such applications a geometric space is disrcetized by a 3d-grid. Then a domain decomposition technique assigns a subdomain to each processor of the network. Each processor performs the computations on mesh points in each subdomain independently. The involved computations in these grids are of two kinds: "dynamics" and "physics". The dynamics calculations correspond to the fluid dynamics of the atmosphere and are applied to the horizontal field. These calculations use explicit schemes to discretize the involved partial differential equations because they are inherently parallel. The physics calculations represent the natural procedures such as clouds, moist convection, the planetary boundary layer and surface processes and are applied to the vertical level. The computations of a vertical column are local, that is, they do not need data from the neighboring columns and are implicit in nature. As the "physics" computations differ from one subdomain to the other the processors have an unbalanced load. In such situations the load has to be balanced. Due to the nature of our problem we will consider iterative algorithms to balance the load. Most of the existing iterative load balancing schemes [12] involve two steps. The first step calculates a balancing flow. This flow is used in the second step, in which load balancing elements are migrated accordingly. It is well known that in distributed systems the communication complexity plays an important role. Therefore, a load balancing algorithm should have a minimum flow and use local communication. Iterative load balancing methods have these characteristics and this is the reason for making them appropriate in our case study. Two main categories are the Diffusion [1, 4], and the Dimension Exchange [4, 24] algorithms. The performance of a balancing algorithm can be measured in terms of number of iterations it requires to reach a balanced state and in terms of the amount of load moved over the edges of the underlying processor graph. In [6] it is shown that all local iterative diffusion schemes calculate a minimal flow with respect to a weighted l_2-norm. In the Diffusion (DF) method [4, 1] a processor simultaneously sends workload to its neighbors with lighter workload and receives from its neighbors with heavier workload. It is assumed that the system is synchronous, homogeneous and the network connections are of unbounded capacity. Under the synchronous assumption, the Diffusion (DF) method has been proved to converge in polynomial time for any initial workload [1].

In this paper we study the application of the DF method for the efficient parallelization of a simple model. Our model involves the numerical solution of the steady state Navier-Stokes (NS) equations in the horizontal plane and random load values, corresponding to the "physics" computations, in the vertical plane. Our intention is to study the performance of a load balancing scheme under more realistic conditions than in previous cases [13, 14]. Up to now the problem of load balancing has been studied solely, namely without any combination of horizontal and vertical computations. In this paper (i) we use an optimal version of the DF method and (ii) the migration of load transfer is implemented using only local communication as opposed to schedules which require global communication.

The paper is organized as follows. In Section 2 we present the Extrapolated Diffusion method, which possesses the fastest rate of convergence among its other counterparts [17]. In Section 3 we describe our simple 'atmospheric model' and we present the use of the Local Modified Successive Overrelaxation (LMSOR) method for the numerical solution of NS equations. In Section 4 we present a domain decomposition technique, which initially assigns a square domain to each processor in a mesh network. However, load balancing imposes a modified domain decomposition. As this increases

*Department of Informatics, University of Athens, Panepistimiopolis, 15784, Athens, Greece ({greg, nmis}@di.uoa.gr).

the communication complexity, we propose a load transfer to balance the load along the row and column processors only. This domain decomposition technique assigns a larger number of processors in the areas of the domain with heavier load, thus increasing the efficiency of load balancing. Finally, in Section 5 we present our simulation results.

2. The Extrapolated Diffusion method. Let us consider an arbitrary, undirected, connected graph $G = (V, E)$. This graph represents our processor network, where its nodes, namely the processors, are denoted by $v_i \in V$ and its edges (links) are $(v_i, v_j) \in E$, when $v_i,\ v_j$ are neighbors. Furthermore, we assign a weight $u_i \geq 0$ to each node, which corresponds to the computational load of v_i. The processor graph reflects the inter-connection of the subdomains of a mesh that has been partitioned and distributed amongst processors (see Figure 4.1). In this graph each node represents a subdomain and two nodes are linked by an edge if the corresponding subdomains share edges of the mesh.

The Extrapolated Diffusion (EDF) method for the load balancing has the form [1, 4]

$$u_i^{(n+1)} = u_i^{(n)} - \tau \sum_{j \in N(i)} c_{ij} \left(u_i^{(n)} - u_j^{(n)} \right), \tag{2.1}$$

where c_{ij} are diffusion parameters, $N(i)$ is the set of the nearest neighbors of node i of the graph $G = (V, E)$, $u_i^{(n)},\ i = 0, 1, 2, \ldots, |V|$ is the load after the n-th iteration on node i and $\tau \in \mathbb{R}\backslash\{0\}$ is a parameter that plays an important role in the convergence of the whole system to the equilibrium state. The overall workload distribution at step n, denoted by $u^{(n)}$, is the transpose of the vector $(u_1^{(n)}, u_2^{(n)}, \ldots, u_{|V|}^{(n)})$ and $u^{(0)}$ is the initial workload distribution. An alternative form of (2.1) is

$$u_i^{(n+1)} = (1 - \tau \sum_{j \in N(i)} c_{ij}) u_i^{(n)} + \tau \sum_{j \in N(i)} c_{ij} u_j^{(n)}, \tag{2.2}$$

which in matrix form becomes

$$u^{(n+1)} = M u^{(n)}, \tag{2.3}$$

where M is called the *diffusion matrix*. From (2.2) it follows that the elements of M, m_{ij}, are equal to τc_{ij}, if $j \in N(i)$, $1 - \tau \sum_{j \in N(i)} c_{ij}$, if $i = j$ and 0 otherwise. With this formulation, the features of diffusive load balancing are fully captured by the iterative process (2.3) governed by the diffusion matrix M. Also, (2.3) can be written as $u^{(n+1)} = (I - \tau L) u^{(n)}$, where $L = BWB^T$ is the *weighted Laplacian* matrix of the graph, W is a diagonal matrix of size $|E| \times |E|$ consisting of the coefficients c_{ij} and B is the vertex-edge incident matrix. At this point, we note that if $\tau = 1$, then we obtain the DF method proposed by Cybenko [4] and Boillat [1], independently. If $W = I$, i. e. if c_{ij}=constant, then we obtain the special case of the DF method with a single parameter τ *(unweighted Laplacian)*. In the unweighted case and for network topologies such as chain, 2D-mesh, nD-mesh, ring, 2D-torus, nD-torus and nD-hypercube, optimal values for the parameter τ that maximize the convergence rate have been derived by Xu and Lau [23, 24]. Recently, we have solved the same problem in its most general form, the weighted case [17]. Next, we present a short review of our results as we use them to further accelerate the rate of convergence of DF.

The diffusion matrix of EDF can be written as

$$M = I - \tau L, \qquad L = D - A, \tag{2.4}$$

where $D = \mathrm{diag}(L)$ and A is the weighted adjacency matrix. Because of (2.4), (2.3) becomes $u^{(n+1)} = (I - \tau D)\, u^{(n)} + \tau A u^{(n)}$ which is the matrix form of (2.2).

The diffusion matrix M must have the following properties: nonnegative, symmetric and stochastic [4, 1]. The eigenvalues of L are $0 = \lambda_1 < \lambda_2 \leq \ldots \leq \lambda_{|V|}$. In case $c_{ij} =$ constant, the optimum value of τ is attained at [22, 25]

$$\tau_o = \frac{2}{\lambda_2 + \lambda_{|V|}}$$

and the corresponding minimum value of the convergence factor

$$\gamma(M) = \max\{|1 - \tau\lambda_{|V|}|, |1 - \tau\lambda_2|\}$$

is given by

$$\gamma_o(M) = \frac{P(L) - 1}{P(L) + 1}, \text{ where } P(L) = \frac{\lambda_{|V|}}{\lambda_2},$$

TABLE 2.1
E=Even, O=Odd, Formulae for the optimum τ_o and $\gamma_o(M)$

N_1	N_2	τ_o	$\gamma_o(M)$
E	E	$\left(3+2\sigma_2-\cos\frac{2\pi}{N_1}\right)^{-1}$	$\frac{1+2\sigma_2+\cos\frac{2\pi}{N_1}}{3+2\sigma_2-\cos\frac{2\pi}{N_1}}$
O	O	$\left(2+\sigma_2(1+\cos\frac{\pi}{N_2})+\cos\frac{\pi}{N_1}-\cos\frac{2\pi}{N_1}\right)^{-1}$	$\frac{\cos\frac{\pi}{N_1}+\cos\frac{2\pi}{N_1}+\sigma_2(1+\cos\frac{\pi}{N_2})}{2+\sigma_2(1+\cos\frac{\pi}{N_2})+\cos\frac{\pi}{N_1}-\cos\frac{2\pi}{N_1}}$
E	O	$\left(3-\cos\frac{2\pi}{N_1}+\sigma_2(1+\cos\frac{\pi}{N_2})\right)^{-1}$	$\frac{1+\cos\frac{2\pi}{N_1}+\sigma_2(1+\cos\frac{\pi}{N_2})}{3-\cos\frac{2\pi}{N_1}+\sigma_2(1+\cos\frac{\pi}{N_2})}$
O	E	$\left(2+2\sigma_2+\cos\frac{\pi}{N_1}-\cos\frac{2\pi}{N_1}\right)^{-1}$	$\frac{2\sigma_2+\cos\frac{\pi}{N_1}-\cos\frac{2\pi}{N_1}}{2+2\sigma_2+\cos\frac{\pi}{N_1}-\cos\frac{2\pi}{N_1}}$

which is the P-condition number of L. Note that if $P(L)\gg 1$, then the rate of convergence of the EDF method is given by

$$R(M)=-\log\gamma_o(M)\simeq\frac{2}{P(L)},$$

which implies that the rate of convergence of the EDF method is a decreasing function of $P(L)$. The problem of determining the diffusion parameters c_{ij} such that EDF attains its maximum rate of convergence is an active research area [5, 9, 17]. Introducing the set of parameters $\tau_i, i=1,2,\ldots,|V|$, instead of a fixed parameter τ in (2.2), the problem moves to the determination of the parameters τ_i in terms of c_{ij}. By considering local Fourier analysis [16, 17] we were able to determine good values (near the optimum) for τ_i. These values become optimum (see Table 2.1) in case the diffusion parameters are constant in each dimension and satisfy the relation $c_j^{(2)}=\sigma_2 c_i^{(1)}$, $i=1,2,\ldots,N_1$, $j=1,2,\ldots,N_2$, where $\sigma_2=\frac{1-\cos\frac{2\pi}{N_1}}{1-\cos\frac{2\pi}{N_2}}$ and $c_i^{(1)}$, $c_j^{(2)}$ are the row and column diffusion parameters, respectively, of the torus [17]. Also, in [17] it is proved that for stretched torus, that is a torus with either $N_1\gg N_2$ or $N_2\gg N_1$,

$$R(EDF)\simeq 2R(DF),$$

at the optimum stage, which means that EDF is twice as fast as DF.

In order to further improve, by an order of magnitude, the rate of convergence of EDF we can apply accelerated techniques (Semi-Iterative, Second-Degree and Variable Extrapolation) following [22, 20, 15].

It is known [22, 25] that the convergence of (2.3) can be greatly accelerated if one uses the Semi-Iterative scheme

$$u^{(n+1)}=\rho_{n+1}(I-\tau_o L)u^{(n)}+(1-\rho_{n+1})u^{(n-1)},$$

with

$$\rho_1=1,\ \rho_2=\left(1-\frac{\sigma^2}{2}\right)^{-1},\ \rho_{n+1}=\left(1-\frac{\sigma^2}{4}\rho_n\right)^{-1},\ n=2,3,\ldots,$$

and

$$\sigma=\gamma_o(M). \tag{2.5}$$

It is worth noting that σ is equal to $\gamma_o(M)$, which is the minimum value of the convergence factor of EDF. In addition, $\gamma_o(M)$ and τ_o, for EDF, are given by the expressions of Table 2.1 for the corresponding values of N_1 and N_2. It can be shown [22, 25] that

$$R(SI-EDF)\simeq\sqrt{2}R(SI-DF)$$

which indicates that the rate of convergence of SI-EDF will be approximately 40% better than that of SI-DF in case of stretched torus.

3. The Atmospheric model. As a case study we will consider the simulation of a simple atmospheric model. In particular, we will consider the numerical solution of NS equations in the horizontal plane by using a parallel version of the SOR iterative method. In the vertical plane we model the implicit "physics" computations by random load values.

3.1. Discretization of the Navier-Stokes equations. The model problem considered here is that of solving the 2-D incopressible Navier-Stokes (NS) equations. The equations in terms of vorticity Ω and stream function Ψ are

$$\Delta\Psi = -\Omega, \qquad u\frac{\partial\Omega}{\partial x} + v\frac{\partial\Omega}{\partial y} = \frac{1}{Re}\Delta\Omega, \tag{3.1}$$

where Re is the Reynold number of the fluid flow and u, v are the velocity components in the y and x directions, respectively. The velocity components are given in terms of the stream function Ψ by $u = \frac{\partial\Psi}{\partial y}$ and $v = -\frac{\partial\Psi}{\partial x}$. If the computational domain is the unit square, then the boundary conditions for such flow are given by $\Psi = 0$, $\frac{\partial\Psi}{\partial x} = 0$ at $x = 0$ and $x = 1$, $\Psi = 0$, $\frac{\partial\Psi}{\partial y} = 0$ at $y = 0$ and $\Psi = 0$, $\frac{\partial\Psi}{\partial y} = -1$ at $y = 1$. The 5-point discretization of NS equations on a uniform $N \times N$ mesh of size $h = \frac{1}{N}$ leads to

$$\frac{1}{h^2}\left[-4\Psi_{ij} + \Psi_{i-1,j} + \Psi_{i+1,j} + \Psi_{i,j+1} + \Psi_{i,j-1}\right] = \Omega_{ij} \tag{3.2}$$

and

$$\Omega_{ij} = l_{ij}\Omega_{i-1,j} + r_{ij}\Omega_{i+1,j} + t_{ij}\Omega_{i,j+1} + b_{ij}\Omega_{i,j-1}, \tag{3.3}$$

with

$$l_{ij} = \frac{1}{4} + \frac{1}{16}Reu_{ij},\ r_{ij} = \frac{1}{4} - \frac{1}{16}Reu_{ij},$$

$$t_{ij} = \frac{1}{4} - \frac{1}{16}Rev_{ij},\ b_{ij} = \frac{1}{4} + \frac{1}{16}Rev_{ij},$$

where

$$u_{ij} = \Psi_{i,j+1} - \Psi_{i,j-1},\ v_{ij} = \Psi_{i-1,j} - \Psi_{i+1,j}.$$

The boundary conditions are also given by

$$\Omega_{i,0} - \frac{2}{h^2}\Psi_{i,1} = 0,\ \Omega_{N,j} - \frac{2}{h^2}\Psi_{N-1,j} = 0$$

and

$$\Omega_{0,j} - \frac{2}{h^2}\Psi_{1,j} = 0,\ \Omega_{i,N} - \frac{2}{h^2}(h - \Psi_{i,N-1}) = 0.$$

3.2. The Local Modified SOR method. The local SOR method was introduced by Ehrlich [7, 8] and Botta and Veldman [2] in an attempt to further increase the rate of convergence of SOR. The idea is based on letting the relaxation factor ω vary from equation to equation. Kuo et. al [19] combined local SOR with red black ordering and showed that is suitable for parallel implementation on mesh connected processor arrays. In the present study we generalize local SOR by letting two different sets of parameters $\omega_{ij}, \omega_{ij}^{'}$ to be used for the red ($i+j$ even) and black ($i+j$ odd) points, respectively. An application of our method to (3.2) and (3.3) can be written as follows

$$\Omega_{ij}^{(n+1)} = (1-\omega_{ij})\Omega_{ij}^{(n)} + \omega_{ij}J_{ij}\Omega_{ij}^{(n)}, \qquad i+j \text{ even}$$

$$\Omega_{ij}^{(n+1)} = (1-\omega_{ij}^{'})\Omega_{ij}^{(n)} + \omega_{ij}^{'}J_{ij}\Omega_{ij}^{(n+1)}, \qquad i+j \text{ odd}$$

with

$$J_{ij}\Omega_{ij}^{(n)} = l_{ij}^{(n)}\Omega_{i-1,j}^{(n)} + r_{ij}^{(n)}\Omega_{i+1,j}^{(n)} + t_{ij}^{(n)}\Omega_{i,j+1}^{(n)} + b_{ij}^{(n)}\Omega_{i,j-1}^{(n)}$$

and

$$l_{ij}^{(n)} = \frac{1}{4} + \frac{1}{16} Reu_{ij}^{(n)},\ r_{ij}^{(n)} = \frac{1}{4} - \frac{1}{16} Reu_{ij}^{(n)},$$

$$t_{ij}^{(n)} = \frac{1}{4} - \frac{1}{16} Rev_{ij}^{(n)},\ b_{ij}^{(n)} = \frac{1}{4} + \frac{1}{16} Rev_{ij}^{(n)},$$

where

$$u_{ij}^{(n)} = \Psi_{i,j+1}^{(n)} - \Psi_{i,j-1}^{(n)}, v_{ij} = \Psi_{i-1,j}^{(n)} - \Psi_{i+1,j}^{(n)}.$$

A similar iterative scheme holds also for Ψ_{ij}. If μ_{ij} are real, then the optimum values of the LMSOR parameters are given by [3]

$$\omega_{1,i,j} = \frac{2}{1 - \overline{\mu}_{ij}\underline{\mu}_{ij} + \sqrt{(1-\overline{\mu}_{ij}^2)(1-\underline{\mu}_{ij}^2)}},\ \omega_{2,i,j} = \frac{2}{1 + \overline{\mu}_{ij}\underline{\mu}_{ij} + \sqrt{(1-\overline{\mu}_{ij}^2)(1-\underline{\mu}_{ij}^2)}},$$

where $\overline{\mu}_{ij}$ and $\underline{\mu}_{ij}$ are computed by

$$\overline{\mu}_{ij} = 2\left(\sqrt{\ell_{ij} r_{ij}}\cos\pi h + \sqrt{t_{ij} b_{ij}}\cos\pi k\right),$$

$$\underline{\mu}_{ij} = 2\left(\sqrt{\ell_{ij} r_{ij}}\cos\frac{\pi(1-h)}{2} + \sqrt{t_{ij} b_{ij}}\cos\frac{\pi(1-k)}{2}\right),$$

with $h = k = \frac{1}{\sqrt{N}}$. If μ_{ij} are imaginary, then the optimum values of the LMSOR parameters are given by

$$\omega_{1,i,j} = \frac{2}{1 - \overline{\mu}_{ij}\underline{\mu}_{ij} + \sqrt{(1+\overline{\mu}_{ij}^2)(1+\underline{\mu}_{ij}^2)}},\ \omega_{2,i,j} = \frac{2}{1 + \overline{\mu}_{ij}\underline{\mu}_{ij} + \sqrt{(1+\overline{\mu}_{ij}^2)(1+\underline{\mu}_{ij}^2)}}.$$

If μ_{ij} are complex of the form $\mu_{ij} = \mu_{Rij} + i\mu_{Iij}$, we use the following heuristic formulas [2]

$$\omega_{1,i,j} = \frac{2}{1 + \overline{\mu}_R\underline{\mu}_R - \overline{\mu}_I\underline{\mu}_I(1 - (\underline{\mu}_R\overline{\mu}_R)^{2/3})^{-1} + \sqrt{M_{R,I}}}$$

and

$$\omega_{2,i,j} = \frac{2}{1 - \overline{\mu}_R\underline{\mu}_R + \overline{\mu}_I\underline{\mu}_I(1 - (\underline{\mu}_R\overline{\mu}_R)^{2/3})^{-1} + \sqrt{M_{R,I}}},$$

where

$$M_{R,I} = [1 - \overline{\mu}_R^2 + \overline{\mu}_I^2(1 - \overline{\mu}_R^{2/3})^{-1}][1 - \underline{\mu}_R^2 + \underline{\mu}_I^2(1 - \underline{\mu}_R^{2/3})^{-1}].$$

4. Domain Decomposition and Load Transfer. Let us assume the domain for the solution of the NS equations is rectangular. Initially, we apply a domain decomposition technique which divides the original domain into p square subdomains, where p is the number of available processors (see Figure 4.1). This decomposition proved to be optimal, in case the load is the same on all mesh points [3], in the sense of minimizing the ratio communication over computation. Next, each subdomain is assigned to a processor in a mesh network. The parallel efficiency depends on two factors: an equal distribution of computational load on the processors and a small communication overhead achieved by minimizing the boundary length. If the latter is achieved by requiring the minimization of the number of cut edges i. e. the total interface length, the mesh partitioning problem turns out to be NP-complete. Fortunately, a number of graph partitioning heuristics have been developed (see e.g. [6, 10]). Most of them try to minimize the cut size which is sufficient for many applications but this approach has also its limitations [11]. To avoid these limitations we apply a load balancing scheme such as to maintain the structure of the original decomposition. This is achieved by adjusting, according to a simple averaging load rule, the width of each row/column. Although this approach cannot achieve full balance as it moves

a constant number of columns and rows, it proves to be efficient. Next, we consider a load balancing scheme, which employs the above feature. Let us assume the situation as illustrated on the left of Figure 4.1. The shaded area denotes the physics computations, i. e. the load distribution. In an attempt to balance the load among the processors we decompose the load area into smaller domains as this is illustrated on the right of Figure 4.1. We will refer to this partitioning as "nesting". The advantage of nesting is that the structure of the domain decomposition graph remains unchanged, thus minimizing the interprocessor communication at the cost of imbalance. The problem now is to determine the width of each row and column. Let us consider the case presented in Figure 4.2, where we have four processors a, b, c, d, each one assigned initially a square with the same area. Further, we assume that the result of the EDF algorithm is that processor a must

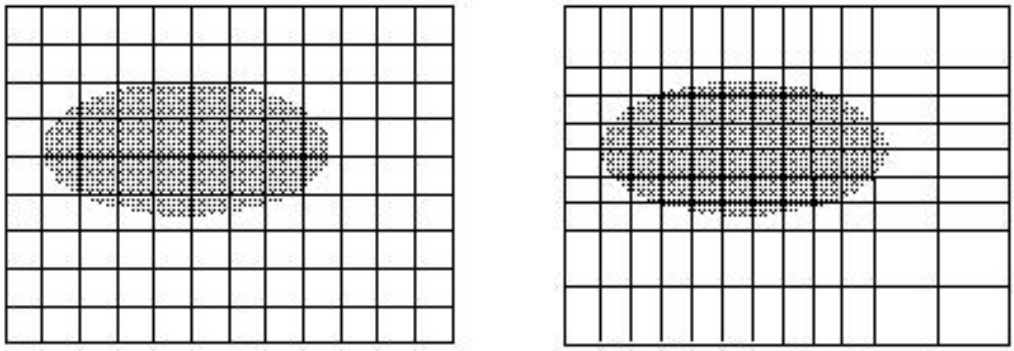

FIG. 4.1. *Domain decomposition by "nesting"*

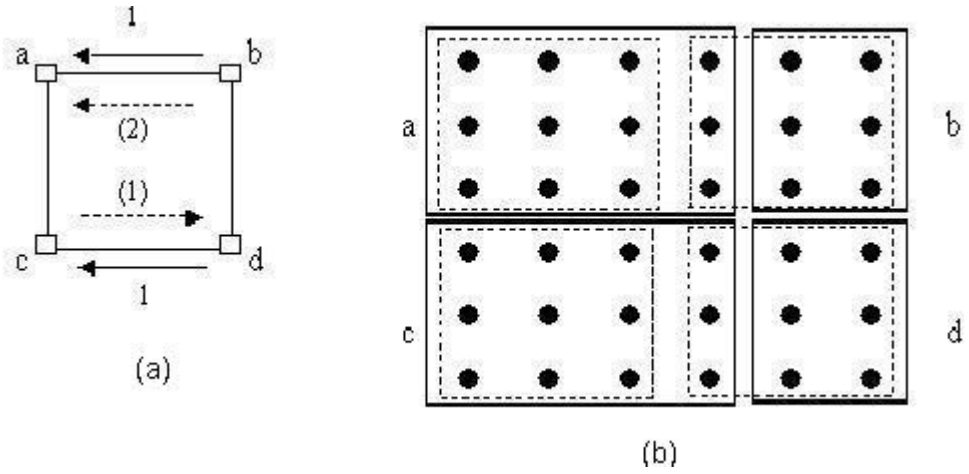

FIG. 4.2. *Column transfer according to the 'weighted average' method*

receive two columns of mesh points from processor b and processor c must send one column to processor d (Figure 4.2(a) dotted arrows). But if these transfers are carried out, they will destroy the mesh structure of the domain decomposition graph as now processor d will have two north neighbors a and b instead of one. In order to avoid this phenomenon we allow the 'weighted average' number of columns to be transferred among the processors a, b and c, d, where 'weighted average'= $\lceil \frac{2+(-1)}{2} \rceil = 1$. This means that processor a will receive one column (instead of two) and processor c will also receive one column (instead of sending one). After the load transfer is carried out, the domain graph remains a mesh as is depicted in Figure 4.2(b) with the solid lines, whereas the dotted lines indicate the initial boundaries of the processors. This process requires communication between processors along two successive rows of the mesh network of processors. A similar procedure for the processors along the columns will fix the width of each column. For a $\sqrt{p} \times \sqrt{p}$ mesh processor network this process requires a total of $O(\sqrt{p})$ communication.

5. Simulation Results. To evaluate the effectiveness of our load balancing algorithm, we ran some tests for the considered model using 44 processors of the HP Superdome parallel machine of the University of Athens. In these tests we compare the behavior of our model against the use of the load balancing algorithm. The method used for the load balancing was the SI-EDF [18]. For the physics computations we assumed a normal distribution of loads superimposed on the given mesh network for solving the NS equations. In particular, we considered that the artificial load is produced by the following function

$$f(x) = Me^{-(\frac{2(x-x_0)}{d})^2},$$

where x denotes the position of a processor in a row of the mesh network, M is the height and d is the width of the distribution (see figure 5.1). In our experiments we kept d constant. Clearly, by varying M we were able to examine the behavior of our load balancing algorithm in different scenarios between the "physics" and "dynamics" calculations.

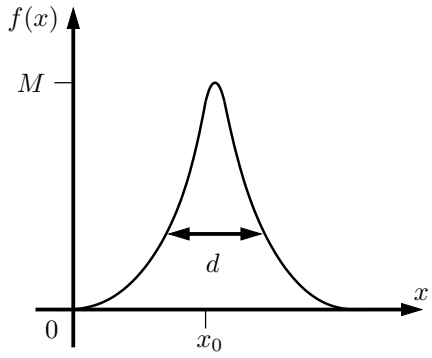

FIG. 5.1. *Random load values*

Our results are summarized in Table 5.1, where we considered different orders for the horizontal mesh as shown in the second column. For each mesh we find the improvement achieved by using our load balancing scheme. The numbers in the 3rd and 4th columns are in secs. By increasing the order of the mesh the rows and columns for each subdomain are also increased. As a consequence, the computation and communication time of LMSOR is also increased. This is clearly observed in both columns (With LB, Without LB) of Table 5.1. Increasing M has the effect of increasing the load in the normal distribution. For small M ($M = 1$) the improvement, by using load balancing, is constant and approximately 21% for meshes with order larger than 40. This was expected as for light load there will be a small number of row/column movement which will be approximately the same for sufficiently large meshes. Also, the overall gain is moderate. For small meshes the horizontal computation time is not large enough to justify the use of load balancing resulting in negative improvement. For heavier load ($M = 100$) the gain in time, due to load balancing, will be greater as is shown in Table 5.1, where we have an improvement of approximately 45% for larger meshes. However, if the load is increased considerably there will also be an increase in load balancing time as more columns and rows need to be moved according to our load balancing algorithm. This has the effect of reducing the improvement. Indeed, if $M = 1000$, then the improvement is less than 45%. From the above it is clear that when the vertical load is small in comparison to the horizontal then load balancing should be avoided. On the other hand, as the load increases, we may reach an improvement of nearly 45% when using the load balancing algorithm. Even though we kept the structure of the application graph unchanged at the cost of approximate balance, the gain is satisfactory.

6. Acknowledgement. The project is co-funded by the European Social Fund and National Resources (EPEAK II) Pythagoras, Grant No. 70/3/7418.

REFERENCES

[1] Boillat J. E., *Load balancing and poisson equation in a graph* , Concurrency: Practice and Experience 2, 1990, 289–313.
[2] Botta E. F. and Veldman A. E. P., *On local relaxation methods and their application to convection-diffusion equations*, J. Comput. Phys., 48, 1981, 127–149.
[3] Boukas L.A. and Missirlis N. M., *The Parallel Local Modified SOR for nonsymmetric linear systems*, Inter. J. Computer Math., 68, 1998, 153–174.
[4] Cybenko G., *Dynamic load balancing for distributed memory multi-processors*, Journal of Parallel and Distributed Computing, 7, 1989, 279–301.
[5] Diekmann R., Muthukrishnan S. and Nayakkankuppam M. V., *Engineering diffusive load balancing algorithms using experiments*. In G. Bilardi et al., editor, IRREGULAR'97, LNCS 1253, 1997, 111–122.

TABLE 5.1
Simulation Results. **LB** =*Load Balancing. Numbers in 3nd and 4th columns are in secs. The last column shows the percentage in improvement.*

Scale Factor	**Mesh Order**	**44 Processors**		
		Without LB	**With LB**	**Improvement**
$M = 1$	20×20	46.932	50.551	-7.71%
	40×40	149.88	117.306	21.73%
	80×80	366.119	287.374	21.51%
	100×100	499.268	394.327	21.02%
$M = 100$	20×20	4689.879	4652.724	0.79%
	40×40	15517.46	9654.508	37.78%
	80×80	37866.911	20745.52	45.21%
	100×100	48499.712	26434.305	45.50%
$M = 1000$	20×20	489042.981	461727.89	5.59%
	40×40	1499122.957	1112865.135	25.77%
	80×80	3767658.518	2186112.869	41.98%
	100×100	4700380.338	2834764.494	30.69%

[6] Diekmann R., Frommer A. and Monien B., *Efficient schemes for nearest neighbour load balancing*, Parallel Computing, 25, 1999, 789–812.
[7] Ehrlich L. W., *An Ad-Hoc SOR Method*, J. Comput. Phys., 42, 1981, 31–45.
[8] Ehrlich L. W., *The Ad-Hoc SOR method: A local relaxation scheme*, in elliptic Problem Solvers II, Academic Press, New York, 1984, 257–269.
[9] Elsässer R., Monien B., Schamberger S. and Rote G., *Optimal Diffusion Schemes and Load Balancing for Product Graphs* , Parallel Proc. Letters, 14, No.1, 2002, 61–73.
[10] Farhat C., Maman N. and Brown G., *Mesh partitioning for implicit computations via iterative domain decomposition: Impact and optimization of the subdomain aspect ratio*, Int. J. Numer. Meth. in Eng., 38, 1995, 989–1000.
[11] Hendrickson B. and Leland R., *An improved spectral graph partitioning algorithm for mapping parallel computations*, SIAM J. Sci. Comput., 16, 1995, 452–469.
[12] Hu Y. F. and Blake R. J., *An improved diffusion algorithm for dynamic load balancing*, Parallel Computing, 25, 1999, 417–444.
[13] Karagiorgos G., Missirlis N. M. and Tzaferis F., *Dynamic Load Balancing for Atmospheric Models*, Proceedings of the Ninth ECMWF Workshop on the use of High Performance Computing in Meteorology, Developments in teracomputing, November 13-17, 2000, Reading, UK, edit. W. Zwieflhofer, N. Kreitz, World Scientific, 214–226.
[14] Karagiorgos G. and Missirlis N. M., *Iterative Load Balancing Schemes for Air Pollution Models in Large-Scale Scientific Computing*, Lecture Notes in Computer Science 2179, S. Margenov, J. Wasniewski and P. Yalamov, Third International Conference, Sozopol, Bulgaria, 2001, 291–298.
[15] Karagiorgos G. and Missirlis N. M., *Accelerated diffusion algorithms for dynamic load balancing*, Information Processing Letters, 84, 2002, 61–67.
[16] Karagiorgos G. and Missirlis N. M., *Fourier analysis for solving the load balancing problem*, Foundations of Computing and Decision Sciences, 27, No 3, 2002, 129–140.
[17] Karagiorgos G. and Missirlis N. M., *Convergence of the diffusion method for weighted torus graphs using Fourier analysis*, Theoretical Computer Science (accepted).
[18] Karagiorgos G., Missirlis N. M. and Tzaferis F., *Fast diffusion load balancing algorithms on torus graphs*, Proceedings of EuroPar 06, Dresden, Germany, August 2006, 222–231.
[19] Kuo C.-C. J., Levy B. C. and Musicus B. R., *A local relaxation method for solving elliptic PDE's on mesh-connected arrays*, SIAM J. Sci. Statist. Comput. 8, 1987, 530–573.
[20] Muthukrishnan S. , Ghosh B. and Schultz M. H. , *First and second order Diffusive methods for rapid, coarse, distributed load balancing*, Theory of Computing Systems, 31, 1998, 331–354.
[21] Ostromsky Tz. and Zlatev Z., *Parallel Implementation of a Large-Scale 3-D Air Pollution Model*, Lecture Notes in Computer Science 2179, S. Margenov, J. Wasniewski and P. Yalamov, Third International Conference, Sozopol, Bulgaria, 2001, 309–316.
[22] Varga R. S., *Matrix Iterative Analysis*, Englewood Cliffs, NJ, Prentice-Hall, 1962.
[23] Xu C. Z. and Lau F. C. M., *Optimal parameters for load balancing the diffusion method in k-ary n-cube networks*, Information Processing Letters 4, 1993, 181–187.
[24] Xu C. Z. and Lau F. C. M., *Load balancing in parallel computers: Theory and Practice*, Kluwer Academic Publishers, Dordrecht, 1997.
[25] Young D. M., *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

Edited by: Svetozar Margenov
Received: Sept 17, 2007
Accepted: Oct 29, 2007