Load Balancing for the Numerical Solution of the Navier-Stokes Equations

Gregory Karagiorgos\textsuperscript{1}, Petros Katsafados\textsuperscript{3}, Andreas Kontarinis\textsuperscript{2}, Nikolaos M. Missirlis\textsuperscript{1}, and Filippos Tzaferis\textsuperscript{1}

\textsuperscript{1} Department of Informatics, University of Athens, Panepistimiopolis, GR-15784, Athens, Greece
\{greg, nmis, ftzaf\}@di.uoa.gr
http://parallel.di.uoa.gr

\textsuperscript{2} ankont@mycosmos.com

\textsuperscript{3} Department of Physics, University of Athens, Panepistimiopolis, GR-15784, Athens, Greece
pkatsaf@mg.uoa.gr
http://parallel.di.uoa.gr

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. It 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 \(\omega\) 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. With our balancing scheme, a gain of approximately 45\% in execution time is achieved, in certain cases.

1 Introduction

In this paper we study the application of the Diffusion method for the efficient parallelization of a simple model. It 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 studies \cite{12,13}. Up to now the problem of load balancing has been studied solely, namely without any combination of horizontal and vertical computations. In this paper it is the first time, where (i) an optimal version of the diffusion method is used and (ii) the migration of
load transfer is also 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 (EDF) method, which possesses the fastest rate of convergence among its other counterparts \[13\]. 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 describe 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 the communication complexity, we propose a load transfer to balance the load along the row 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 (EDF) 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 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 \[14\]

\[
\begin{align*}
    u_i^{(n+1)} &= u_i^{(n)} - \tau \sum_{j \in A(i)} c_{ij} \left( u_i^{(n)} - u_j^{(n)} \right) \\
    \text{where } c_{ij} \text{ are diffusion parameters, } A(i) \text{ is the set of the nearest neighbors of node } i \text{ of the graph } G = (V, E), u_i^{(n)}, \, i = 0, 1, 2, \ldots, |V| \text{ is the load after the } n\text{-th iteration on node } i \text{ and } \tau \in \mathbb{R} \setminus \{0\} \text{ is a parameter that plays an important role in the convergence of the whole system to the equilibrium state. The overall workload distribution at step } n \text{, denoted by } u^{(n)}, \text{ is the transpose of the vector } (u_1^{(n)}, u_2^{(n)}, \ldots, u_{|V|}^{(n)}) \text{ and } u^{(0)} \text{ is the initial workload distribution. In matrix form } (1) \text{ becomes}
\end{align*}
\]

\[
\begin{align*}
    u^{(n+1)} &= M u^{(n)} \\
    \text{where } M \text{ is called the diffusion matrix. The elements of } M, m_{ij}, \text{ are equal to } \tau c_{ij}, \text{ if } j \in A(i), \, 1 - \tau \sum_{j \in A(i)} c_{ij}, \text{ if } i = j \text{ and } 0 \text{ otherwise. With this formulation, the features of diffusive load balancing are fully captured by the iterative process (2) governed by the diffusion matrix } M. \text{ Also, (2) can be written as } u^{(n+1)} = (I - \tau L) u^{(n)}, \text{ where } L = BW^T B^T \text{ is the weighted Laplacian matrix of the graph, } W \text{ is a diagonal matrix of size } |E| \times |E| \text{ consisting of the coefficients}
\end{align*}
\]
\( c_{ij} \) and \( B \) is the vertex-edge incident matrix. At this point, we note that if \( \tau = 1 \), then we obtain the Diffusion (DF) method proposed by Cybenko [4] and Boillat [1], independently. If \( W = I \), then we obtain the special case of the DF method with a single parameter \( \tau \) (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 \( \tau \) that maximize the convergence rate have been derived by Xu and Lau [21,22]. However, the same problem, in the weighted case was solved recently [16]. Next, we consider the weighted case. The diffusion matrix of EDF can be written as

\[
M = I - \tau L, \quad L = D - A \quad (3)
\]

where \( D = \text{diag}(L) \) and \( A \) is the weighted adjacency matrix. Because of (3), (2) becomes \( u^{(n+1)} = (I - \tau D) u^{(n)} + \tau Au^{(n)} \) or in component form

\[
u_i^{(n+1)} = \left(1 - \tau \sum_{j \in A(i)} c_{ij}\right) u_i^{(n)} + \tau \sum_{j \in A(i)} c_{ij} u_j^{(n)}, \quad i = 1, 2, \ldots, |V| \quad (4)
\]

The diffusion matrix \( M \) must have the following properties: nonnegative, symmetric and stochastic [4,11]. The eigenvalues of \( L \) are \( 0 = \lambda_1 < \lambda_2 \leq \ldots \leq \lambda_n \). In case \( c_{ij} = \) constant, the optimum value of \( \tau \) is attained at \( \tau_o = \frac{2}{\lambda_2 + \lambda_n} \)

and the corresponding minimum value of the convergence factor

\[
\gamma(M) = \max\{|1 - \tau \lambda_n|, |1 - \tau \lambda_2|\}
\]

is given by

\[
\gamma_o(M) = \frac{P(L) - 1}{P(L) + 1}, \quad \text{where} \quad P(L) = \frac{\lambda_n}{\lambda_2}
\]

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 [5,9,16]. Introducing the set of parameters \( \tau_i, i = 1, 2, \ldots, |V| \), instead of a fixed parameter \( \tau \) in [11], the problem moves to the determination of the parameters \( \tau_i \) in terms of \( c_{ij} \). By considering local Fourier analysis [15,16] we were able to determine good values (near the optimum) for \( \tau_i \). These values become optimum(see Table[11] in case the diffusion parameters are constant in each dimension and satisfy the relation \( c_{ij}^{(2)} = \sigma_2 c_{ij}^{(1)}, \quad i = 1, 2, \ldots, N_1, \quad j = 1, 2, \ldots, N_2 \),
Table 1. Formulae for the optimum $\tau_0$ and $\gamma_o(M)$ ( E: Even, O: Odd )

<table>
<thead>
<tr>
<th>$N_1$</th>
<th>$N_2$</th>
<th>Case</th>
<th>$\tau_0$</th>
<th>$\gamma_o(M)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>E</td>
<td>1</td>
<td>$(3 + 2\sigma - \cos \frac{2\pi}{N_1})^{-1}$</td>
<td>$\frac{1 + 2\sigma + \cos \frac{2\pi}{N_1}}{3 + 2\sigma - \cos \frac{2\pi}{N_1}}$</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>2</td>
<td>$(2 + \sigma(1 + \cos \frac{\pi}{N_2}) + \cos \frac{\pi}{N_1} - \cos \frac{2\pi}{N_1})^{-1}$</td>
<td>$\frac{\cos \frac{\pi}{N_1} + \cos \frac{2\pi}{N_1} + \sigma (1 + \cos \frac{\pi}{N_2})}{2 + \sigma (1 + \cos \frac{\pi}{N_2}) + \cos \frac{\pi}{N_1} - \cos \frac{2\pi}{N_1}}$</td>
</tr>
<tr>
<td>E</td>
<td>O</td>
<td>3</td>
<td>$(3 - \cos \frac{2\pi}{N_1} + \sigma (1 + \cos \frac{\pi}{N_2}))^{-1}$</td>
<td>$\frac{1 + \cos \frac{2\pi}{N_1} + \sigma (1 + \cos \frac{\pi}{N_2})}{3 - \cos \frac{2\pi}{N_1} + \sigma (1 + \cos \frac{\pi}{N_2})}$</td>
</tr>
<tr>
<td>O</td>
<td>E</td>
<td>4</td>
<td>$(2 + 2\sigma + \cos \frac{\pi}{N_1} - \cos \frac{2\pi}{N_1})^{-1}$</td>
<td>$\frac{2\sigma + \cos \frac{\pi}{N_1} - \cos \frac{2\pi}{N_1}}{2 + 2\sigma + \cos \frac{\pi}{N_1} - \cos \frac{2\pi}{N_1}}$</td>
</tr>
</tbody>
</table>

where $\sigma = \frac{1 - \cos \frac{2\pi}{N_2}}{1 - \cos \frac{2\pi}{N_2}}$ and $c_i^{(1)}, c_j^{(2)}$ are the row and column diffusion parameters, respectively, of the torus [16]. Also, in [16] it is proven that at the optimum stage

$$RR_\infty(EDF) \simeq 2RR_\infty(DF)$$

which means that EDF is twice as fast as DF for stretched torus, that is a torus with either $N_1 \gg N_2$ or $N_2 \gg N_1$.

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 [20,19,14].

### 2.1 The Semi-iterative method

We now consider iterative schemes for further accelerating the convergence of EDF. It is known [20,23] that the convergence of (2) can be greatly accelerated if one uses the Semi-Iterative scheme

$$u^{(n+1)} = \rho_{n+1}(I - \tau_0L)u^{(n)} + (1 - \rho_{n+1})u^{(n-1)}$$

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

$$\sigma = \gamma_o(M)$$

It is worth noting that $\sigma$ is equal to $\gamma_o(M)$, which is the minimum value of the convergence factor of EDF. In addition, $\gamma_o(M)$ and $\tau_0$, for EDF, are given by the expressions of Table 1 for the corresponding values of $N_1$ and $N_2$. It can be shown [20,23] that

$$RR_\infty(SI - EDF) \simeq \frac{1}{\sqrt{2}} RR_\infty(SI - DF)$$

$^1 RR(.) = \frac{1}{R(.)}$. 

---

Load Balancing for the Numerical Solution of the Navier-Stokes Equations 767
which indicates that the number of iterations of SI-EDF will be approximately 30% less than the number of iterations 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. The atmospheric models of climate and weather use a 3-dimensional grid to represent the atmosphere’s behavior. The involved computations in these grids are of two kinds: dynamics and physics [13,12]. 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. Alternatively, 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 far as our simulation study is concerned, we will model the computations in the horizontal plane by solving the Navier-Stokes equations and the computations in the vertical plane with random load values, corresponding to the implicit physics calculations.

3.1 Numerical Solution of the Navier-Stokes (NS) Equations

The model problem considered here is that of solving the 2-D incompressible Navier-Stokes (NS) equations. The equations in terms of vorticity $\Omega$ and stream function $\Psi$ are

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

(6)

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 $\Psi$ 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 $\Psi = 0, \frac{\partial \Psi}{\partial y} = 0$ at $x = 0$ and $x = 1$, $\Psi = 0, \frac{\partial \Psi}{\partial x} = 0$ 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 grid of mesh size $h = 1/N$ leads to

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

(7)

and

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

(8)

with $l_{ij} = 1/4 + 1/16Re u_{ij}$, $r_{ij} = 1/4 - 1/16Re u_{ij}$, $t_{ij} = 1/4 - 1/16Re v_{ij}$, $b_{ij} = 1/4 + 1/16Re v_{ij}$, 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
\[ \Omega_{i,0} - \frac{2}{h^2} \Psi_{i,1} = 0, \quad \Omega_{N,j} - \frac{2}{h^2} \Psi_{N-1,j} = 0 \]
and
\[ \Omega_{0,j} - \frac{2}{h^2} \Psi_{1,j} = 0, \quad \Omega_{i,N} - \frac{2}{h^2} (h - \Psi_{i,N-1}) = 0 \]

### 3.2 The Local Modified SOR Method

The local SOR method was introduced by Ehrlich [7] 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 \( \omega \) vary from equation to equation. Kuo et al [18] 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 \((i + j \text{ even})\) and black \((i + j \text{ odd})\) points, respectively. An application of our method to (7) and (8) can be written as follows

\[
\Omega_{ij}^{(n+1)} = (1 - \omega_{ij}) \Omega_{ij}^{(n)} + \omega_{ij} J_{ij} \Omega_{ij}^{(n)}, \quad i + j \text{ even}
\]
and

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

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 \( l_{ij}^{(n)} = 1/4 + 1/16 \text{Re} u_{ij}^{(n)}, \quad r_{ij}^{(n)} = 1/4 - 1/16 \text{Re} u_{ij}^{(n)}, \quad t_{ij}^{(n)} = 1/4 - 1/16 \text{Re} u_{ij}^{(n)}; \quad b_{ij}^{(n)} = 1/4 + 1/16 \text{Re} u_{ij}^{(n)} \)

where \( u_{ij}^{(n)} \) are complex, we propose the following heuristic formulas

\[
\omega_{1,i,j} = \frac{2}{1 - \overline{\mu}_{ij} \mu_{ij} + \sqrt{(1 - \overline{\mu}_{ij})(1 - \mu_{ij}^2)}}, \quad \omega_{2,i,j} = \frac{2}{1 + \overline{\mu}_{ij} \mu_{ij} + \sqrt{(1 + \overline{\mu}_{ij})(1 + \mu_{ij}^2)}},
\]

where \( \overline{\mu}_{ij} \) and \( \mu_{ij} \) are computed by

\[
\overline{\mu}_{ij} = 2 \left( \sqrt{\ell_{ij} \tau_{ij} \cos \frac{\pi(1-h)}{2}} + \sqrt{\ell_{ij} \beta_{ij} \cos \frac{\pi(1-k)}{2}} \right), \quad \mu_{ij} = 2 \left( \sqrt{\ell_{ij} \tau_{ij} \cos \frac{\pi(1-h)}{2}} + \sqrt{\ell_{ij} \beta_{ij} \cos \frac{\pi(1-k)}{2}} \right),
\]

with \( h = k = 1/\sqrt{N} \).

If \( \mu_{ij} \) are imaginary, then the optimum values of the LMSOR parameters are given by

\[
\omega_{1,i,j} = \frac{2}{1 - \overline{\mu}_{ij} \mu_{ij} + \sqrt{(1 + \overline{\mu}_{ij})(1 + \mu_{ij}^2)}}, \quad \omega_{2,i,j} = \frac{2}{1 + \overline{\mu}_{ij} \mu_{ij} + \sqrt{(1 + \overline{\mu}_{ij})(1 + \mu_{ij}^2)}},
\]

If \( \mu_{ij} = \mu_{R,ij} + i \mu_{I,ij} \) are complex, we propose the following heuristic formulas

\[
\omega_{1,i,j} = \frac{2}{1 + \overline{\mu}_{R} \mu_{R} + \overline{\mu}_{I} \mu_{I} (1 - (\mu_{R} \overline{\mu}_{R})^{2/3})^{-1} + \sqrt{M_{R,I}}}
\]
and

\[
\omega_{2,i,j} = \frac{2}{1 - \overline{\mu}_{R} \mu_{R} + \overline{\mu}_{I} \mu_{I} (1 - (\mu_{R} \overline{\mu}_{R})^{2/3})^{-1} + \sqrt{M_{R,I}}}
\]

where

\[
M_{R,I} = [1 - \overline{\mu}_{R}^2 + \overline{\mu}_{I}^2 (1 - \overline{\mu}_{R}^2)^{2/3} (1 - \mu_{R}^{2/3})^{-1}] [1 - \overline{\mu}_{R}^2 + \overline{\mu}_{I}^2 (1 - \mu_{R}^{2/3})^{-1}]
\]
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 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 reducing or increasing, according to a simple averaging load rule, the width of each row. Although this approach reduces the effectiveness of the method, as will not achieve full balance, 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 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 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 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

Fig. 1. Domain decomposition by nesting
algorithm is that processor $a$ must receive two columns of mesh points from processor $b$ and processor $c$ must send one column to processor $d$ (Figure 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 neighbors $a$ and $b$. 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 $= \left\lceil \frac{2 + (\frac{2}{2})}{2} \right\rceil = 1$. 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 2(b) with the solid lines. 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 this process requires a total $O(\sqrt{p})$ communication.

5 Simulation Results

To evaluate the effectiveness of our load balancing algorithm, we ran some tests for the considered model. In these tests we examined two cases each time: the model running without the load balancing algorithm against to its periodical use for a specific time interval. The method used was the SI-EDF [17] for different mesh sizes. For the physics computations we assumed a normal distribution of loads superimposed on the given mesh for solving the NS equations. In fact, we used a scale factor $M$, which determines the imbalance of computational vertical load and obtained results for different values of this parameter. So, we were able to examine the behavior of our load balancing algorithm in different scenarios between the physics and dynamics calculations. Our results, summarized in Table 2 indicate that when the vertical load is small in relation 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 SI-EDF load bal-
Table 2. Simulation Results. L/B = Load Balancing. Numbers in 2nd and 3rd columns are secs. The last column shows the percentage improvement.

<table>
<thead>
<tr>
<th>mesh size</th>
<th>Without L/B</th>
<th>With L/B</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>M = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20x20</td>
<td>46.932</td>
<td>50.551</td>
<td>−7.71%</td>
</tr>
<tr>
<td>40x40</td>
<td>149.88</td>
<td>117.306</td>
<td>21.73%</td>
</tr>
<tr>
<td>80x80</td>
<td>366.119</td>
<td>287.374</td>
<td>21.51%</td>
</tr>
<tr>
<td>100x100</td>
<td>499.268</td>
<td>394.327</td>
<td>21.02%</td>
</tr>
<tr>
<td>M = 100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20x20</td>
<td>4689.879</td>
<td>4652.724</td>
<td>0.79%</td>
</tr>
<tr>
<td>40x40</td>
<td>15517.46</td>
<td>9654.508</td>
<td>37.78%</td>
</tr>
<tr>
<td>80x80</td>
<td>37866.911</td>
<td>20745.52</td>
<td>45.21%</td>
</tr>
<tr>
<td>100x100</td>
<td>48499.712</td>
<td>26434.305</td>
<td>45.50%</td>
</tr>
<tr>
<td>M = 10000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20x20</td>
<td>489042.981</td>
<td>461727.89</td>
<td>5.59%</td>
</tr>
<tr>
<td>40x40</td>
<td>1499122.957</td>
<td>1112865.135</td>
<td>25.77%</td>
</tr>
<tr>
<td>80x80</td>
<td>3767658.518</td>
<td>2186112.869</td>
<td>41.98%</td>
</tr>
<tr>
<td>100x100</td>
<td>4700380.338</td>
<td>2834764.494</td>
<td>30.69%</td>
</tr>
</tbody>
</table>

The choice of L/B algorithm. Even though we kept the structure of the application graph unchanged with the cost of approximate balance, the gain is satisfactory.

Acknowledgement

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

References