Parallel two-stage algorithms for solving the PageRank problem

Héctor Migallón[a], Violeta Migallón[b,*], José Penadés[b]

[a] *Department of Physics and Computer Architectures, University Miguel Hernández, E-03202 Elche, Alicante, Spain*
[b] *Department of Computer Science and Artificial Intelligence, University of Alicante, E-03071 Alicante, Spain*

Abstract

In this work we present parallel algorithms based on the use of two-stage methods for solving the PageRank problem as a linear system. Different parallel versions of these methods are explored and their convergence properties are analyzed. The parallel implementation has been developed using a mixed MPI/OpenMP model to exploit parallelism beyond a single level. In order to investigate and analyze the proposed parallel algorithms, we have used several realistic large datasets. The numerical results show that the proposed algorithms can speed up the time to converge with respect to the parallel Power algorithm and behave better than other well-known techniques.

Keywords: PageRank, parallel algorithms, two-stage methods, shared memory, distributed memory

1. Introduction

PageRank is one of the most known and influential methods for ranking Web pages [1]. This model, used by the Google search engine, revolutionized Web search by providing a reliable, spam-resistant way to find reputable Web pages. The model is based on two key ideas: first, that links between Web pages provide information about their importance, and second, that the relationship

*Corresponding author. Tel.: +34965903900; fax: +34965903902.
Email address: violeta@ua.es (Violeta Migallón)

between importance and linking is recursive. The PageRank algorithm involves essentially the computation of the dominant eigenvector of a Markov matrix describing the behaviour of a model Web surfer jumping from page to page on the Web hyperlink graph. Originally, the method of choice for computing the PageRank vector was the Power method [2]. However, in the last years, several techniques to accelerate the Power method have been developed such as extrapolation methods [3, 4, 5], adaptive methods [6] or Arnoldi-type algorithms [7, 8, 9]. Taking into account that Brin and Page originally conceived the PageRank problem as an eigenvector problem nearly all research has focused on the eigenvector formulation of the PageRank. However, the normalized eigenvector problem can be solved using its linear system formulation opening new research lines [10]. Recently, some approaches based on linear system formulations have been considered, see e.g., [11, 12, 13, 14, 15, 16] and the references cited therein.

Additionally, due to the large size of the Web link graph, to deal with realistic problems, a promising way of accelerating PageRank is parallel processing. By using the sparse linear system formulation of the PageRank problem, in [12] parallel algorithms for computing PageRank based on Krylov subspace methods are investigated. The analysis of the results shows that the convergence of Krylov methods strongly depends on the graph and although the Krylov methods have the highest average convergence rate and fastest convergence by number of iterations, on some graphs, the parallel running time of these methods can be greater than the running time of the parallel Power method. In [13], inner-outer algorithms are investigated for the dense linear system formulation of the PageRank problem. These inner-outer methods compare favourably with other widely used schemes. Moreover, their parallel implementation achieves a substantial gain with respect to the Power method. An OpenMP code was implemented to work with the matrix stored by either out or in-edges and it works with Web graphs compressed into *bvgraph* data structure [17].

In this work we design new parallel algorithms for computing PageRank. These algorithms use two-stage methods for solving the PageRank problem by means of its sparse linear system formulation. Concretely, in Section 2 we

describe the PageRank problem. In Section 3 we review the stationary two-stage methods with the non-stationary ones. In Section 4 we introduce the proposed methods and we analyze their convergence. Section 5 explains the MPI/OpenMP hybrid parallel implementation of the algorithms. In Section 6 some details about sparse matrix storage formats and data partitioning are given, justifying the approach proposed in this work to solve the PageRank problem. Section 7 explores the behaviour of the proposed parallel two-stage algorithms for computing PageRank using realistic test data on a current Symmetric Multi-Processing (SMP) supercomputer. Finally, in Section 8 we give some conclusions. This paper is based upon Migallón et al. [18], but the current paper includes the following additional research: new strategies for solving the PageRank problem are considered and their convergence analyzed. In addition, new numerical experiments are performed and explored.

2. The PageRank problem

PageRank is a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. The PageRank problem can be seen as a matrix problem. Concretely, let $G = [g_{ij}]_{i,j=1}^{n}$ be a Web graph adjacency matrix with elements $g_{ij} = 1$ when there is a link from page j to page i, with $i \neq j$, and zero otherwise. Here n is the number of Web pages. From this matrix we can construct a transition matrix $P = [p_{ij}]_{i,j=1}^{n}$ as follows: $p_{ij} = \frac{g_{ij}}{c_j}$ if $c_j \neq 0$ and 0 otherwise, where $c_j = \sum_{i=1}^{n} g_{ij}$, $1 \leq j \leq n$, represents the number of out-links from a page j. For pages with a nonzero number of out-links, i.e., $c_j \neq 0$ for all j, $1 \leq j \leq n$, the matrix P is column stochastic. Thus, each element of this matrix has values between 0 and 1, and the sum of the components of each column is 1. In this case the PageRank vector can be obtained by solving $P\pi = \pi$. The Power method [2] is one of the oldest and simplest iterative methods for solving this eigenvector problem. When the nonnegative matrix P is irreducible (i.e., its graph is strongly connected) and stochastic, the Power method converges

to the eigenvector corresponding to $\lambda_{max} = 1$, and when normalized, it is the stationary probability distribution over pages under a random walk on the Web. However, the Web contains many pages without out-links, called dangling nodes. Dangling pages present a problem for the mathematical PageRank formulation because in this case the matrix P is non-stochastic and the Power method can not be used. Moreover, the matrix irreducibility is not satisfied for a Web graph. In order to overcome these difficulties, Page and Brin [1] change the transition matrix P to a column stochastic matrix $\bar{P} = \alpha(P + vd^T) + (1 - \alpha)ve^T$, where $d \in \Re^n$ is the dangling page indicator defined by $d_i = 1$ if and only if $c_i = 0$, $e^T = (1, 1, \ldots, 1)$, and the vector $v \in \Re^n$ is some probability distribution over pages. This model means that the random surfer jumps from a dangling page according to a distribution v. Originally uniform teleportation $v = \frac{e}{n}$ was used. Then, setting α such that $0 < \alpha < 1$, the matrix $\bar{P}$ is column stochastic, irreducible and it preserves the L_1 norm, that is, $\|\bar{P}x\|_1 = \|x\|_1$ and therefore the Power method can be used to solve the stationary distribution of the ergodic Markov chain defined by

$$\bar{P}\pi = \pi. \tag{1}$$

Note that the parameter α is a parameter that controls the proportion of time the random surfer follows the hyperlinks as opposed to teleporting. Even though the parameter α was originally set to 0.85 by Google founders Brin and Page, this parameter can be increased to capture the true essence of the Web, giving less weight to the artificial teleportation matrix. However as $\alpha \to 1$, the Power method takes an increasing amount of time to converge and other techniques need to be experimented.

Let $P' = P + vd^T$, then the normalized eigenvector problem (1) can be rewritten with some algebraic manipulations as $(I - \alpha P')x = (1 - \alpha)v$, with $\pi = \frac{x}{\|x\|_1}$. Using this formulation, researchers have recently experimented with new PageRank techniques [12, 14, 15, 16] such as inner-outer or multisplitting iterations for solving this linear system. However, the matrix $I - \alpha P'$ can be pretty dense, whenever the number of dangling nodes is large because these

completely sparse columns are replaced with completely dense columns. For this reason, it is more suitable to operate on the very sparse matrix P. Following [10], we can write the PageRank problem in terms of the matrix P. That is, let the PageRank vector π be such that $\bar{P}\pi = \pi$, then π can be obtained solving the linear system $(I - \alpha P)x = v$, and letting $\pi = \frac{x}{\|x\|_1}$. Thinking about PageRank as a sparse linear system opens new research lines combining iterative techniques and parallel processing in order to reduce the work associated with the PageRank vector computation. Taking into account the advantage of the two-stage methods [19, 20] for parallel computing, we focus on these iterative schemes for accelerating PageRank computations.

3. Stationary and non-stationary two-stage methods

Consider the problem of solving a linear system

$$Ax = b, \tag{2}$$

where A is an $n \times n$ nonsingular matrix and x and b are n-vectors. Classical iterative methods proceed by solving at each step a simpler linear system induced by a splitting of A into $A = M - N$ (M nonsingular); i.e., beginning with an arbitrary vector $x^{(0)}$, the iteration procedure

$$Mx^{(l+1)} = Nx^{(l)} + b, \quad l = 0, 1, 2, \ldots, \tag{3}$$

is used to compute a sequence of iterate vectors that converges to the solution of (2) if and only if the iteration matrix $M^{-1}N$ is convergent, that is if the spectral radius of the iteration matrix is less than one ($\rho(M^{-1}N) < 1$); see e.g., [21].

On the other hand, when the linear systems (3) are not solved exactly, but rather their solutions approximated by iterative methods, we are in the presence of a two-stage method; see e.g., [22, 23]. That is, consider the splitting $M = F - G$ and perform, at each outer step l, $q(l)$ inner iterations of the iterative procedure induced by this splitting. Thus, the two-stage method can

be written as follows

$$x^{(l+1)} = (F^{-1}G)^{q(l)}x^{(l)} + \sum_{j=0}^{q(l)-1} (F^{-1}G)^j F^{-1}(Nx^{(l)} + b), \quad l = 0, 1, \ldots. \quad (4)$$

When the number of inner iterations stays fixed in each outer step, i.e., when $q(l) = q,\ q \geq 1$, it says that the method is stationary, while a non-stationary two-stage method is such that the number of inner iterations may change with the outer iteration. Obviously, stationary two-stage methods are special cases of non-stationary ones.

Given an initial vector $x^{(0)}$, the two-stage iterative method (4) produces the sequence of vectors

$$x^{(l+1)} = T_{q(l)}x^{(l)} + c_{q(l)}, \quad l = 0, 1, 2, \ldots, \quad (5)$$

where $T_{q(l)} = (F^{-1}G)^{q(l)} + \sum_{j=0}^{q(l)-1}(F^{-1}G)^j F^{-1}N,\ l = 0, 1, 2, \ldots$, are the iteration matrices and $c_{q(l)} = \sum_{j=0}^{q(l)-1}(F^{-1}G)^j F^{-1}b,\ l = 0, 1, 2 \ldots$.

Let ξ be the exact solution of the linear system (2) and let $\epsilon^{(l+1)} = x^{(l+1)} - \xi$ be the error at the $l + 1$ iteration. It is easy to prove that ξ is a fixed point of (5). Thus, $\epsilon^{(l+1)} = T_{q(l)}\epsilon^{(l)} = \cdots = T_{q(l)}T_{q(l-1)} \cdots T_{q(0)}\epsilon^{(0)},\ l = 0, 1, 2, \ldots$. Therefore, for any initial vector $x^{(0)}$ the sequence of vectors generated by iteration (5) converges to the solution of the linear system (2) if and only if $\lim_{l\to\infty} T_{q(l)}T_{q(l-1)} \cdots T_{q(0)} = O$, where O denotes the null matrix. Since, for the stationary case $T_{q(l)} = T_q$, for all $l = 0, 1, 2, \ldots$, then the convergence of (5) is equivalent to requiring $\rho(T_q) < 1$. In [24] it was shown that if both the inner and outer splittings are convergent then the stationary two-stage method converges for large enough q. However, in the non-stationary case, $\rho(T_{q(l)}) < 1,\ l = 0, 1, 2, \ldots$, does not necessarily imply $\lim_{l\to\infty} T_{q(l)}T_{q(l-1)} \cdots T_{q(0)} = O$. Therefore, tools other than the spectral radius are necessary for its analysis. Theorems 1 and 2 give conditions on the inner and outer splittings for convergence of both stationary and non-stationary two-stage methods for any number of inner iterations. First we present some definitions and preliminaries used in the paper. The notation and terminology adopted in this paper are along the lines of those used in [21]. Given a matrix $A = (a_{ij})$ we define $|A| = (|a_{ij}|)$.

Clearly all entries of $|A|$ are nonnegative, that is $|A| \geq O$. A general matrix A is called an M-matrix if A can be expressed as $A = sI - B$, with $B \geq O$, $s > 0$, and $\rho(B) \leq s$. The M-matrix A is singular when $s = \rho(B)$ and nonsingular when $s > \rho(B)$. Let $Z^{n\times n}$ denote the set of all real $n \times n$ matrices which have all non-positive off-diagonal entries. A nonsingular matrix $A \in Z^{n\times n}$ is an M-matrix if and only if A is a monotone matrix ($A^{-1} \geq O$). For any matrix $A = (a_{ij}) \in \Re^{n\times n}$ we define its comparison matrix $\langle A\rangle = (\alpha_{ij}) \in \Re^{n\times n}$ by

$$\alpha_{ii} = |a_{ii}|, \quad \alpha_{ij} = -|a_{ij}|, \ i \neq j.$$

A nonsingular matrix A is said to be an H-matrix if $\langle A\rangle$ is an M-matrix. Of course, M-matrices are special cases of H-matrices.

Definition 1. [21, 22] *Let $A \in \Re^{n\times n}$. A splitting $A = M - N$ is called*
(a) regular if $M^{-1} \geq O$ and $N \geq O$,
(b) weak regular if $M^{-1} \geq O$ and $M^{-1}N \geq O$,
(c) H-splitting if $\langle M\rangle - |N|$ is an M-matrix, and
(d) H-compatible splitting if $\langle A\rangle = \langle M\rangle - |N|$.

Lemma 1. [21] *Let $A = M - N$ be a splitting. If the splitting is weak regular then $\rho(M^{-1}N) < 1$ if and only if $A^{-1} \geq O$.*

Theorem 1. [22] *Let $A = M - N$ be a convergent regular splitting, and let $M = F - G$ be a convergent weak regular splitting. Then, the two-stage iterative method (4) converges to the solution of the nonsingular linear system $Ax = b$, for any initial vector $x^{(0)}$ and for any sequence of inner iterations $q(l) \geq 1$, $l = 0, 1 \ldots$.*

Theorem 2. [22] *Let $A = M - N$ be an H-splitting , and let $M = F - G$ be an H-compatible splitting. Then, the two-stage iterative method (4) converges to the solution of the nonsingular linear system $Ax = b$, for any initial vector $x^{(0)}$ and for any sequence of inner iterations $q(l) \geq 1$, $l = 0, 1 \ldots$.*

4. Two-stage methods for solving PageRank

The PageRank vector π such that $\bar{P}\pi = \pi$ can be obtained solving the sparse linear system

$$(I - \alpha P)x = v, \tag{6}$$

and letting $\pi = \frac{x}{\|x\|_1}$; see e.g., [10].

For this purpose, we first propose the use of the iteration scheme (4) for solving the linear system (6), where the outer splitting is $(I - \alpha P) = M - N$, with $M = I - \beta P$, and $N = (\alpha - \beta)P$ and the inner splitting is $M = F - G$, with $F = I$ and $G = \beta P$, $\beta \in \Re$. That is, given an initial vector $x^{(0)}$ the proposed two-stage iterative method produces the sequence of vectors

$$x^{(l+1)} = T_{q(l)}x^{(l)} + c_{q(l)}, \quad l = 0, 1, 2, \ldots, \tag{7}$$

where

$$T_{q(l)} = (\beta P)^{q(l)} + (\alpha - \beta) \sum_{j=0}^{q(l)-1} (\beta P)^j P, \quad l = 0, 1, 2, \ldots,$$

and $c_{q(l)} = \sum_{j=0}^{q(l)-1} (\beta P)^j v,\ l = 0, 1, 2 \ldots.$

To reduce the number of outer iterations of a two-stage method sometimes a relaxation parameter $\omega > 0$ is introduced in the inner iterations. For our problem, this technique really consists of replacing the inner splitting by $I - \beta P = \bar{F} - \bar{G}$, with $\bar{F} = \frac{1}{\omega} I$ and $\bar{G} = \frac{1-\omega}{\omega} I + \beta P$ and produces the following iterative scheme.

$$x^{(l+1)} = \bar{T}_{q(l)}x^{(l)} + \bar{c}_{q(l)}, \quad l = 0, 1, 2, \ldots, \tag{8}$$

where

$$\bar{T}_{q(l)} = ((1-\omega)I + \omega\beta P)^{q(l)} + \omega(\alpha - \beta) \sum_{j=0}^{q(l)-1} ((1-\omega)I + \omega\beta P)^j P, \quad l = 0, 1, 2, \ldots,$$

and $\bar{c}_{q(l)} = \sum_{j=0}^{q(l)-1} ((1-\omega)I + \omega\beta P)^j v,\ l = 0, 1, 2 \ldots.$

Clearly with $\omega = 1$ the two-stage method (7) is recovered. In the case of $\omega \neq 1$ a relaxed two-stage method is obtained. In what follows, we analyze the

convergence of the two-stage iterative method (7) and its relaxed version (8) to solve the PageRank problem.

Theorem 3. *Let P be the transition matrix of the PageRank problem and consider its linear system formulation (6). Let α and β be such that $0 < \beta < \alpha < 1$.*

(i) The two-stage iterative method (7) converges to the solution of the linear system (6), for any initial vector $x^{(0)}$ and for any sequence of inner iterations $q(l) \geq 1,\ l = 0, 1 \ldots$.

(ii) If in addition $0 < \omega < 1$, the theorem holds for the relaxed two-stage iterative method (8).

Proof. Let $I - \alpha P = M - N$, with $M = I - \beta P$, and $N = (\alpha - \beta)P$. Since $0 < \beta < \alpha < 1$, the matrices $I - \alpha P$ and $I - \beta P$ are nonsingular matrices, $(I - \alpha P)^{-1} \geq O$ and $(I - \beta P)^{-1} \geq O$. Moreover $(\alpha - \beta)P \geq O$. Therefore, the splitting $I - \alpha P = M - N$ is a regular splitting and from Lemma 1 it follows that it is a convergent regular splitting. Taking into account that $M = I - \beta P = F - G$, with $F = I$ and $G = \beta P$ is a weak regular splitting and $(I - \beta P)^{-1} \geq O$, using Lemma 1 and Theorem 1 the proof of *(i)* is completed.

On the other hand, when $0 < \omega < 1$, the splitting $M = I - \beta P = \bar{F} - \bar{G}$, with $\bar{F} = \frac{1}{\omega} I$ and $\bar{G} = \frac{1-\omega}{\omega} I + \beta P$ is a regular splitting , and therefore the proof of *(ii)* follows in the same way as the proof of *(i)*. ∎

Theorem 4. *Let P be the transition matrix of the PageRank problem and consider its linear system formulation (6). Let α and β be such that $0 < \alpha < 1$ and $0 < \alpha < \beta < \frac{1+\alpha}{2}$.*

(i) The two-stage iterative method (7) converges to the solution of the linear system (6), for any initial vector $x^{(0)}$ and for any sequence of inner iterations $q(l) \geq 1,\ l = 0, 1 \ldots$.

(ii) If in addition $0 < \omega < 1$, the theorem holds for the relaxed two-stage iterative method (8).

Proof. Let $I-\alpha P = M-N$, with $M = I-\beta P$, and $N = (\alpha-\beta)P$. Since $0 < \alpha < \beta < \frac{1+\alpha}{2}$, we can write $\langle M\rangle - |N| = \langle I-\beta P\rangle - |(\alpha-\beta)P| = I-\beta P-(\beta-\alpha)P = I-(2\beta-\alpha)P$ and taking into account that $0 < 2\beta-\alpha < 2\frac{1+\alpha}{2}-\alpha = 1$, the matrix $I-(2\beta-\alpha)P$ is nonsingular and $(I-(2\beta-\alpha)P)^{-1} \geq O$. That is $\langle M\rangle - |N|$ is an M-matrix. Then, from Definition 1 it follows that $I-\alpha P = M-N$ is an H-splitting. Taking into account that $\langle I-\beta P\rangle = \langle I\rangle - |\beta P|$, the inner splitting $M = I-\beta P = F-G$, with $F = I$ and $G = \beta P$ is an H-compatible splitting and using Theorem 2 the proof of *(i)* is completed.

On the other hand, if $0 < \omega < 1$ it yields $\langle I-\beta P\rangle = I-\beta P = \langle \frac{1}{\omega} I\rangle - |\frac{1-\omega}{\omega} I + \beta P|$, then the splitting $M = I-\beta P = \bar{F}-\bar{G}$, with $\bar{F} = \frac{1}{\omega} I$ and $\bar{G} = \frac{1-\omega}{\omega} I + \beta P$ is an H-compatible splitting and the proof of *(ii)* follows in the same way as the proof of *(i)*. ■

Example 1 shows that the hypotheses of Theorem 4 on the parameter β cannot be weakened. The calculations of this example have been performed in MATLAB.

Example 1. Consider the linear system (6), with

$$P = \begin{bmatrix} 0 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1 \\ 0 & 0 & 1/3 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{bmatrix},$$

Setting $\alpha = 0.4$ and $\beta = 0.71 > \frac{1+\alpha}{2} = 0.7$ in the iterative scheme (7), the iteration matrix of the stationary two-stage method $T_{q(l)}$, with $q(l) = q \geq 10$ has spectral radius greater than 1, and so it is not convergent.

As we explained in Section 2, two-stage methods can also be used to solve the PageRank problem using the following linear system formulation

$$(I-\alpha P^{'})x = (1-\alpha)v, \tag{9}$$

where $P^{'} = P + vd^T$. In this case, setting the outer splitting $I - \alpha P^{'} = M - N$, with $M = I - \beta P'$, and $N = (\alpha - \beta)P'$ and the inner splitting is $M = F - G$, with $F = I$ and $G = \beta P'$, the two-stage iterative scheme produces the sequence of vectors

$$x^{(l+1)} = \tilde{T}_{q(l)} x^{(l)} + \tilde{c}_{q(l)}, \quad l = 0, 1, 2, \ldots, \tag{10}$$

where

$$\tilde{T}_{q(l)} = (\beta(P + vd^T))^{q(l)} + (\alpha - \beta) \sum_{j=0}^{q(l)-1} (\beta(P + vd^T))^j (P + vd^T), \quad l = 0, 1, 2, \ldots,$$

and $\tilde{c}_{q(l)} = (1 - \alpha) \sum_{j=0}^{q(l)-1} (\beta(P + vd^T))^j (1 - \alpha)v, \ l = 0, 1, 2 \ldots.$

The iteration scheme (10) can be seen as a generalization of the basic inner-outer method proposed in [13], in which $0 < \beta < \alpha < 1$ and the number of inner iterations to performing at each outer iteration was obtained by means of an inner tolerance. In [13] convergence of the inner and outer iterations is analyzed separately, assuming that the inner system is solved exactly. However, when the inner iterations are solved inexactly we are in the presence of a non-stationary two-stage method and only the convergence of the outer and inner splittings does not assure the global convergence of the method. We want to remark that the global convergence of the iterative scheme (10) can be assured for α and β satisfying the same conditions of Theorems 3 and 4; see Theorem 5.

Theorem 5. *Let P be the transition matrix of the PageRank problem and consider its linear system formulation (9), with $P^{'} = P + vd^T$. Let $0 < \alpha < 1$ and $\beta > 0$ be such that $\beta < \alpha$ or $\alpha < \beta < \frac{1+\alpha}{2}$. Then, the two-stage iterative method (10) converges to the solution of the linear system (9), for any initial vector $x^{(0)}$ and for any sequence of inner iterations $q(l) \geq 1, \ l = 0, 1 \ldots.$*

Proof. Consider the outer splitting $I - \alpha P^{'} = M - N$, with $M = I - \beta P^{'}$, and $N = (\alpha - \beta)P^{'}$ and the inner spitting $M = I - \beta P^{'} = F - G$, with $F = I$ and $G = \beta P'$. Taking into account the properties of the matrices $I - \alpha P^{'}$ and $I - \beta P^{'}$ the proof follows reasoning in a similar way as in Theorems 3 and 4. ■

Although the matrices $I-\alpha P$ and $I-\alpha P^{'}$ have similar properties, the matrix $I-\alpha P^{'}$ can be too dense, depending on the number of dangling nodes. However, the matrix $I-\alpha P$ is a very sparse matrix, because a large proportion of its elements are zero. This is due to most webpages link to only a handful of other pages. Sparse matrices such as $I-\alpha P$ are welcome for parallel processing. First, they require significantly less storage since compressed sparse matrix storage formats can be used while very large matrices are infeasible to manipulate the algorithms using dense-matrix structures. Second, matrix-vector product involving a sparse matrix requires much less effort than the dense computation.

5. Parallel algorithms: programming models and hardware

In order to design the parallel two-stage algorithm for solving the PageRank problem, we consider without loss of generality that the transition matrix P is partitioned into p row blocks of the form

$$P = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_p \end{bmatrix},$$

where each P_i, $1 \leq i \leq p$, is a matrix of order $n_i \times n$, with $\sum_{i=1}^{p} n_i = n$. Analogously, we consider the iterate vectors $x^{(l)}$ and v partitioned according to the block structure of P. Then, Algorithm 1 describes the proposed parallel algorithm to solve the sparse linear system (6) by means of the two-stage iterative method (7). In what follows, we have labeled it as the LTW algorithm. Note that, the LTW algorithm allows us to reduce the number of global iterations by eliminating synchronization points at which a process must wait for information from other processes being that each part of $x^{(l+1)}$ is updated more than once ($q(l) > 1$ times) without waiting for the other parts of $x^{(l+1)}$. Note that when $q(l) = 1$, the synchronization among processes is performed at each lth outer iteration in such a way that all processes use the same current

vector $x^{(l)}$ to compute their part of the following iterate vector. In this case no synchronizations and communications among processes are avoided.

A possible way to reduce the number of outer iterations of a two-stage method consists of introducing a relaxation parameter ω in Algorithm 1 adding after the computation of (11) the equation $y_i^{(k)} = \omega y_i^{(k)} + (1-\omega) y_i^{(k-1)}$. In this case we obtain a parallel relaxed two-stage algorithm, that we have called RTS algorithm.

All parallel algorithms analyzed in this work have been implemented in C++ on an HPC cluster of 26 nodes HP Proliant SL390s G7 connected through a network of low-latency QDR Infiniband-based. Each node consists of two Intel XEON X5660 hexacore at up to 2.8 GHz and 12MB cache per processor, with 48 GB of RAM. The operating system is CentOS Linux 5.6 for x86 64 bit. Taking into account the hierarchical hardware design of this high performance system, the parallel algorithms have been implemented combining distributed memory parallelization on the node interconnect with shared memory parallelization inside each node. For this purpose, MPI [25] and OpenMP [26] programming models were combined into a hybrid paradigm in which MPI is used for data distribution among nodes and OpenMP to exploit loop level parallelism within each node. In this way we have used a philosophy of distributed shared memory (DSM) in which $s \times c$ indicates that s nodes of the parallel platform have been used for data distribution and for each one of these nodes, c OpenMP threads have been considered. Particularly, if $s = 1$, the algorithms are executed in shared memory (SM) using c threads on a single node. Conversely, if $c = 1$, we are working on distributed memory (DM) using s nodes.

In order to explain more specifically our approach we can consider Algorithm 1 describes the parallelization of the two-stage method using a distributed memory programming. In this algorithm, the value of p indicates the number of MPI processes, each of them assigned to a physical node. In the distributed shared memory programming, an MPI/OpenMP hybrid parallel algorithm is considered in which every MPI process, mapped into a multicore node, spawns a number of OpenMP threads to execute in parallel the computation of (11),

Algorithm 1: Parallel two-stage method for solving PageRank (LTW algorithm).

Initialization $x^{(0)} = \frac{e}{n}, v = \frac{e}{n},\ q(l),\ l = 1, 2, \cdots,\ k = 0,\ l = 0;$

for $i = 1, 2, \ldots, p$, **do in parallel**

  repeat

    $l = l + 1;$

    $y^{(0)} = x^{(l-1)};$

    for $k = 1, 2, \ldots, q(l)$, **do**

$$\begin{cases} y_i^{(k)} = \beta P_i y^{(k-1)} + (\alpha - \beta) P_i x^{(l-1)} + v_i; \\ y_j^{(k)} = y_j^{(k-1)},\ j \neq i\ ; \end{cases} \tag{11}$$

    end

    $x_i^{(l)} = y_i^{(q(l))};$

    Perform an all-gather operation to obtain $x^{(l)} = [x_1^{(l)}, \ldots, x_p^{(l)}];$

    Compute $\|x_i^{(l)} - x_i^{(l-1)}\|_1;$ **(12)**

    Perform a sum all-to-all reduction over $\|x_i^{(l)} - x_i^{(l-1)}\|_1$ to obtain $\delta = \|x^{(l)} - x^{(l-1)}\|_1;$

  until $\delta < \epsilon;$

  Compute $\|x_i^{(l)}\|_1;$ **(13)**

  Perform a sum all-to-all reduction over $\|x_i^{(l)}\|_1$ to obtain $\|x^{(l)}\|_1;$

  Compute $\pi_i = \frac{x_i^{(l)}}{\|x^{(l)}\|_1};$

  Perform a gather operation over π_i to obtain $\pi = \frac{x^{(l)}}{\|x^{(l)}\|_1}$ in a root process;

end

Algorithm 2: Multicore parallel computation of (11).

for $k = 1, 2, \ldots, q(l) - 1$, **do**
  #pragma omp parallel for schedule(dynamic)
  for $j = startRow, \ldots, endRow$ **do**
    $y_i[j]^{(k)} = \beta P_i[j] y^{(k-1)} + (\alpha - \beta) P_i[j] x^{(l-1)} + v_i[j];$
  end
end

Algorithm 3: Multicore parallel computation of (11) and (12).

$k = q(l);$
#pragma omp parallel for schedule(dynamic) reduction(+:norm)
for $j = startRow, \ldots, endRow$ **do**
  $y_i[j]^{(k)} = \beta P_i[j] y^{(k-1)} + (\alpha - \beta) P_i[j] x^{(l-1)} + v_i[j];$
  $norm+ = fabs(y_i[j]^{(k)} - x_i[j]^{(l-1)});$
end

(12) and (13). These three tasks are parallelized using OpenMP by heading the corresponding "for" loop with a "parallel for" pragma ("#pragma omp parallel for") that launches multiple threads that execute the loop iterations in some system-specific scheme. The computation of (11) was partitioned using a dynamic scheduling strategy, where groups of a user determined size (called chunk size) are assigned to the threads on a first-come, first-served basis. Algorithm 2 describes this scheme in which each iteration j of the parallel "for" corresponds to the computation of the jth component of the vector $y_i^{(k)}$ (denoted as $y_i[j]^{(k)}$) associated to the jth row of the matrix block P_i (denoted as $P_i[j]$). In our case, this strategy is preferable than a static scheduling scheme in which groups of consecutive iterations are assigned to the threads in a round-robin fashion. As we will see later, a static scheme can lead to an unbalanced computational cost since the number of nonzero elements in the matrix P would be very different from one thread to another. On the other hand, no noticeable differences have

been found depending on the chunk size value used in the scheduling of the OpenMP parallel loops, except if very low or high values are used.

The computation of (12) and (13) includes a reduction process at the end of the OpenMP parallel "for". In both computations, the static scheduling strategy should be a priori a good strategy because, in these cases, the computational work is balanced among threads. However, in order to reduce the number of fork/join concurrency control mechanisms, we have included the computation of (12) in the same OpenMP parallel "for" that computes (11) at the last inner iteration, that is, when $k = q(l)$. More specifically, we perform $q(l)-1$ OpenMP parallel loops "for" without a reduction procedure, shown in Algorithm 2, and one last OpenMP parallel loop "for" with a reduction procedure to compute additionally the norm $\|x_i^{(l)} - x_i^{(l-1)}\|_1$, shown in Algorithm 3; note that $y_i^{(q(l))}$ in the last inner iteration corresponds to $x_i^{(l)}$.

6. Data structures and partitioning

There are a multitude of sparse matrix representations, each one with different storage requirements, computational characteristics, and methods of accessing and manipulating matrix entries [27], [28], [29]. The Compressed Sparse Row (CSR) format for sparse matrices is perhaps the most widely used format when no assumptions about the sparsity structure of the matrix is required. This format stores the matrix using three vectors: one for floating point numbers and other two for integers. The floating point vector stores the values of the nonzero elements of the matrix, following a row-wise method. One of the integer vectors stores the column indexes of the elements in the values vector. The other integer vector stores the locations in the values vector that start a row. Another well-known sparse matrix storage format is the $ELLPACK$ format. This format stores the sparse matrix on two arrays, one for floating point numbers, to store the nonzero elements, and one for integers, to store the columns of every nonzero element. On the other hand, blocking storage formats based on the CSR format such as the Block Compressed Sparse row ($BCSR$)

or the Variable Block Row (VBR) formats could be considered.

Comparisons of these storage formats and other basic formats have been reported by different authors, see e.g. [27], [30], [31], [32], and the references cited therein. In [30] the sparse matrix-vector multiplication is analyzed on a wide class of matrices and on a variety of superscalar architectures, showing that the use of the CSR format performs better than other basic formats including the $ELLPACK$ format. Since the $ELLPACK$ format uses the same number of elements per row padding with zero elements, if the percentage of zeros is high and there is a very irregular location of entries in different rows, then the performance of the $ELLPACK$ data structure decreases and storage requirements increase [31]. In [32] a comparative study of block based storage methods for sparse matrix-vector multiplication on multicore architectures is performed. The analysis includes the $BCSR$ and VBR formats and shows that when improper blocks are selected, a dramatic performance degradation is observed compared with the use of the CSR format. That is, the performance of a sparse matrix storage format depends considerably on the nonzero pattern, the block sizes, and the underlying micro-architectures.

Considering the above-mentioned, in this work, a modified Compressed Sparse Row ($mCSR$) format has been used to store the matrices. We represent the two vectors of indexes of the CSR format by integers without sign of 32 bits, while the values and the iterate vectors are represented by means of double precision floating point with 64 bits. Taking into account that, for each column of a transition matrix, all nonzero elements are equal to a fixed value, the $mCSR$ format stores once these elements in an ordered vector. For the experiments performed in this work, we have used four datasets of different sizes, available from http://law.di.unimi.it [33]; see Table 1. These transition matrices have been generated from a web-crawl [34]. The proposed $mCSR$ format has involved a reduction of memory requirements for these matrices of about $63 - 73\%$ with respect to the original CSR format [35]. Table 2 compares both the sequential and parallel Power algorithms, using this $mCSR$ format and the *bvgraph* data structure used in [12] for solving the PageRank problem. As it can

be seen, the *bvgraph* data structure does not provide an efficient way to store the Web graph matrix. In fact, this data structure has caused $4-8x$ slowdown with respect to the $mCSR$ storage format. Therefore, all algorithms treated here have been implemented to work with the matrix stored using the $mCSR$ format.

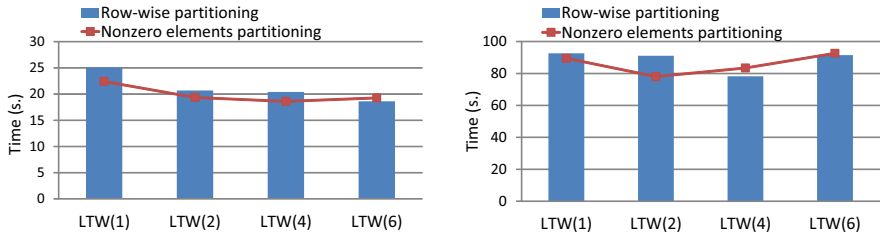
Graph	n	nnz	Dgn (%)	Dens	M (GB)
it-2004	41,291,594	1,150,725,436	12.76	27.87	4.75
webbase-2001	118,142,155	1,019,903,190	23.41	8.63	5.12
uk-2006-10	93,436,772	3,130,910,405	13.52	33.50	12.71
uk-2007-05	105,896,555	3,738,733,648	12.23	35.31	15.11

Table 1: Graphs collection. n =number of nodes, nnz =number of arcs, Dgn=percentage of dangling nodes, Dens=density (arcs/nodes), M=memory requirements using $mCSR$ format.

		Using *bvgraph* (hours)		Using $mCSR$ (hours)	
Graph	Iter.	Sequential	Parallel	Sequential	Parallel
it-2004	869	3.79	1.05	0.61	0.21
webbase-2001	904	6.83	2.16	0.93	0.38
uk-2006-10	761	7.89	2.73	1.50	0.56
uk-2007-05	745	9.38	2.96	1.89	0.67

Table 2: Execution time of the sequential and parallel Power methods, SM (1×4), *bvgraph* versus $mCSR$ format, $\alpha = 0.99$, $\epsilon = 10^{-6}$.

Implementing the two-stage methods for solving PageRank in a parallel environment opens several possibilities of data partitioning and load balancing. For this purpose, the use of hypergraph partitioning-based decomposition techniques was analyzed in [36] applied to the parallel computation of the PageRank vector by means of the Power method using relatively small matrices. Both one and two-dimensional hypergraph decomposition models were considered using the parallel hypergraph partitioner tool Parkway 2.0 [37]. The results show



(a) $\alpha = 0.85$, $\beta = 0.84$, it-2004 matrix.

(b) $\alpha = 0.9$, $\beta = 0.8$, uk-2006-10 matrix.

Figure 1: Row-wise versus nonzero elements distribution, parallel LTW algorithm, DSM (4×4), $\epsilon = 10^{-6}$.

that hypergraph-based partitioning can reduce the communication volume over conventional partitioning schemes, however due to the high initial partitioning overhead, it seems a not plausible approach for large Web graph transition matrices.

# MPI Proc.	MPI rank	# rows	Percentage	# nonzero elements	Percentage
1	0	41291594	100.0%	1150725436	100.0%
2	0	20645797	50.0%	539947754	46.9%
	1	20645797	50.0%	610777682	53.1%
4	0	10322899	25.0%	239969086	20.9%
	1	10322899	25.0%	299978669	26.1%
	2	10322899	25.0%	285580832	24.8%
	3	10322897	25.0%	325196849	28.3%
8	0	5161450	12.5%	111962381	9.7%
	1	5161450	12.5%	128006708	11.1%
	2	5161450	12.5%	140826365	12.2%
	3	5161450	12.5%	159152303	13.8%
	4	5161450	12.5%	145685174	12.7%
	5	5161450	12.5%	139895659	12.2%
	6	5161450	12.5%	138425034	12.0%
	7	5161444	12.5%	186771812	16.2%

Table 3: Workload assigned to each MPI process, row-wise partitioning, it-2004 matrix.

In this work, we have considered two different methods for partitioning the

# MPI Proc.	MPI rank	# rows	Percentage	# nonzero elements	Percentage
1	0	41291594	100.0%	1150725436	100.00%
2	0	21944911	53.1%	575366136	50.0%
	1	19346683	46.9%	575359300	50.0%
4	0	12303064	29.8%	287681937	25.0%
	1	9641847	23.4%	287684199	25.0%
	2	10218835	24.7%	287682404	25.0%
	3	9127848	22.1%	287676896	25.0%
8	0	6498561	15.7%	143841795	12.5%
	1	5804506	14.1%	143840795	12.5%
	2	4661921	11.3%	143840689	12.5%
	3	4979923	12.1%	143842857	12.5%
	4	5231933	12.7%	143841010	12.5%
	5	4986902	12.1%	143841394	12.5%
	6	5325390	12.9%	143840837	12.5%
	7	3802458	9.2%	143836059	12.5%

Table 4: Workload assigned to each MPI process, nonzero elements partitioning, it-2004 matrix.

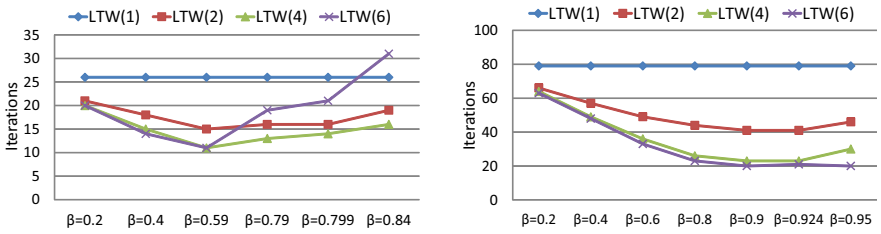
link matrix among nodes. The first method is a row-wise partitioning where each node gets the same amount of rows. The second one is a nonzero elements partitioning in which each node has to handle approximately the same amount of nonzero elements. Figure 1 compares these strategies of data distribution. The notation LTW(q) indicates that, for every l, $q(l) = q$ steps are used in Algorithm 1. Generally, the nonzero elements partitioning has been the best partitioning strategy not only for the LTW algorithm but also for the Power algorithm. This is due to the fact that the row-wise partitioning of a sparse matrix may lead to a load imbalance among processes when the number of nonzero entries of each block of rows differs immensely. However, the nonzero partitioning approach has achieved good load balancing among processes since all processes terminate each global iteration at approximately the same time. Tables 3 and 4 display, for the it-2004 matrix, the number of rows and the number of nonzero elements assigned to each MPI process when row-wise partitioning and nonzero

elements partitioning are used, respectively. As it can be seen, the workload (associated to the number of nonzero elements) becomes unbalanced when row-wise partitioning is used. For example, when 8 MPI processes are used, the process with MPI rank equal to 0 has associated a 9.7% of the workload, while the process with rank 7 has associated a 16.2% of the workload. Conversely, using the nonzero elements partitioning, all MPI processes have associated a 12.5% of the workload, but obviously different number of rows. Taking into account the best performance of the nonzero elements partitioning approach, in what follows, we will afford to work only with this data distribution strategy.

7. Experimental analysis

In this section we analyze the behaviour of the proposed parallel two-stage algorithms for computing PageRank. For this purpose, we have run our algorithm for several values of α, β and ω. In the experiments reported here, we have used for the stopping criterion $\epsilon = 10^{-6}$. Furthermore, the chunk size used in the OpenMP parallel loops was set to 1000.

Figure 2 illustrates the convergence rates for the LTW method, setting a global convergence scheme and different values of β. The weights given to the Web link graph in this figure have been $\alpha = 0.6$ (Figure 2(a)) and $\alpha = 0.85$ (Figure 2(b)). That is, in Figure 2(a), the model assumes that the 60% of the time the random surfer follows the hyperlink structure of the Web and the other 40% of the time he teleports to a random new page. In Figure 2(b) these percentages are 85% and 15%, respectively. Note that good choices of the parameter β can be found when the convergence is assured from a theoretical point of view, that is setting $0 < \beta < \alpha < 1$, or $0 < \alpha < 1$ and $\alpha < \beta < \frac{1+\alpha}{2}$ (see Theorems 3 and 4). Our experience indicates that for setting optimal splittings in the LTW algorithm one must choose β close to α and less than $\frac{1+\alpha}{2}$. In fact, the number of iterations starts to decrease as β increases up to optimal values of β close to that interval. However, when $\beta > \frac{1+\alpha}{2}$ the number of iterations starts to increase and even the convergence may not be guaranteed.



(a) $\alpha = 0.6$, $\frac{1+\alpha}{2} = 0.8$.

(b) $\alpha = 0.85$, $\frac{1+\alpha}{2} = 0.925$.

Figure 2: Number of iterations of the parallel LTW algorithm, DSM (4×4), uk-2006 matrix.

Due to the fact that large values of α give more weight to the true link structure of the Web it is more desirable to choose α close to one. Nevertheless, we want to point out that each choice of α corresponds to a different PageRank problem. In light of this, Table 5 compares the top 50 Web pages returned by the LTW algorithm using $\alpha = 0.85$ with other values of α. Both, the number of common Web pages (# cp) and the number of Web pages located in the same position (# sp) are displayed. As it can be seen, the PageRank vector is very sensitive to changes in α for these Web graphs.

	it-2004		webbase-2001		uk-2006-10		uk-2007-05	
α	# cp	# sp	# cp	# sp	# cp	# sp	# cp	# sp
0.5	40	9	28	0	29	6	28	0
0.6	42	7	29	1	32	7	32	0
0.7	45	10	31	2	40	12	36	1
0.8	47	21	43	3	46	14	43	8
0.9	47	12	46	3	45	2	47	5
0.95	35	7	38	1	39	1	37	4
0.99	8	0	17	1	17	0	19	0
0.995	4	0	9	1	15	0	17	0

Table 5: Comparison of the top 50 Web pages for several values of α in relation to $\alpha = 0.85$.

Table 6 summarizes the effect of α on the expected number of iterations for our datasets by means of their median. We also display the confidence intervals

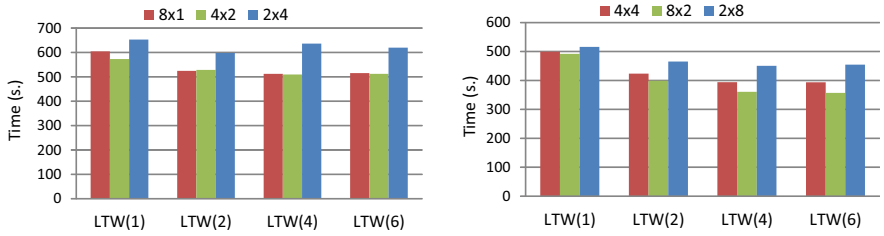
		Iterations	% Reduction	
α	Method	Median	Mean	CI (95%)
$\alpha = 0.85$	LTW(2)	41.5	28.77	[21.01, 36.52]
	LTW(4)	23	60.79	[57.88, 63.69]
	LTW(6)	16	72.72	[70.70, 74.74]
$\alpha = 0.9$	LTW(2)	63.5	23.71	[21.26, 26.16]
	LTW(4)	34.5	58.55	[56.73, 60.37]
	LTW(6)	24	71.17	[70.64, 71.70]
$\alpha = 0.95$	LTW(2)	127.5	25.92	[17.07, 34.76]
	LTW(4)	68	60.49	[55.86, 65.13]
	LTW(6)	48	72.26	[69.11, 75.41]

Table 6: Effect of α in the number of iterations of the LTW algorithms with respect to the Power algorithm, $\beta = \alpha - 0.01$, SM (1×2).

(CI) for the percentage of reduction in the number of iterations in relation to the Power method. These intervals have been obtained using a significance level of 0.05. Taking into account that the sample is small, checking the assumption of normality is needed. For all cases, the Shapiro-Wilk test does not reject the assumption of normality. As it can be seen, the LTW algorithms reduce the number of iterations in relation to the Power method. Then, a good choice for the value of q is one which balances the realization of more local updates with the decrease of the global iterations.

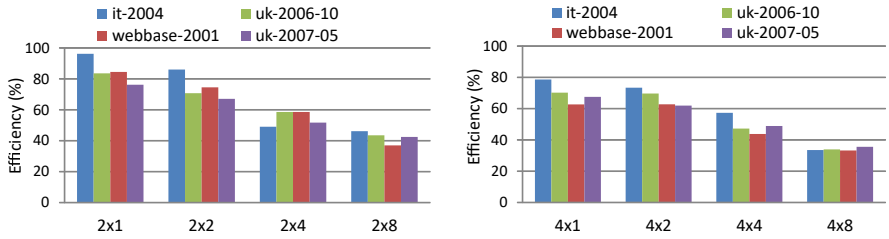
In Figure 3 we illustrate the performance of the LTW algorithm for several values of q, setting $\alpha = 0.98$. In each subfigure, the total number of processes remains unchanged (8 and 16 for Figures 3(a) and 3(b), respectively) but varying the DSM configuration. The best results have been obtained using $q = 2$ or $q = 4$ inner iterations at each global iteration of the LTW algorithm outperforming the LTW(1) algorithm in which synchronization points among processes are not avoided.

Figures 4 and 5 display the efficiency achieved setting, as sequential reference



(a) uk-2006-10. (b) uk-2007-05.

Figure 3: Total running times of the parallel LTW algorithm, $\alpha = 0.98$, $\beta = 0.97$.



(a) DSM $(2 \times c)$. (b) DSM $(4 \times c)$.

Figure 4: Efficiency, Power method as sequential reference algorithm, $\alpha = 0.98$, $\beta = 0.97$, $q = 4$.

algorithms, the Power method and the best LTW algorithm, respectively. We obtain an acceptable scalability being generally quite efficient to use no more than 8 threads per each MPI process. This is due to the fact that many processing cores sharing the same system bus and memory bandwidth can limit the real performance advantage.

On the other hand, we have compared the LTW algorithm with the well-known extrapolation methods [4] and with both the basic inner-outer and the inner-outer Power iterative algorithms proposed in [13]. For this purpose, these algorithms have been implemented in parallel using the same storage format as that used for the LTW and Power algorithms, that is the $mCSR$ format. Furthermore, the parallel implementation of these methods has also been developed using a hybrid MPI/OpenMP scheme. We point out that the performance

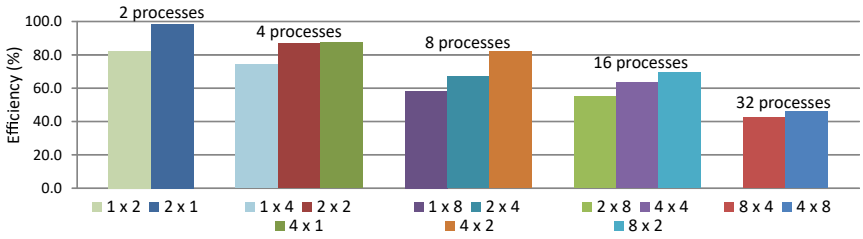
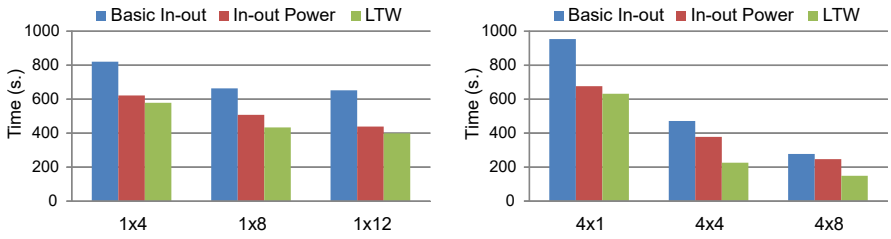


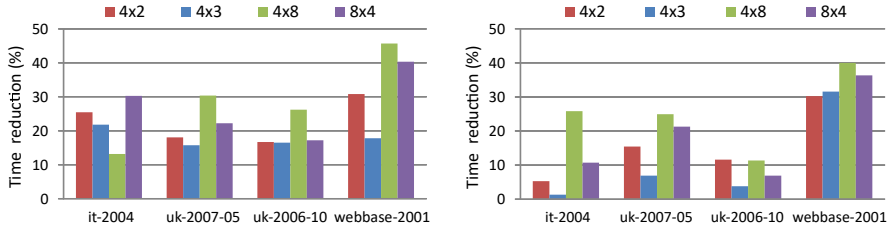
Figure 5: Efficiency, LTW as sequential reference algorithm, $\alpha = 0.98$, $\beta = 0.97$, uk-2007-05 matrix.



(a) SM ($1 \times c$)

(b) DSM ($4 \times c$)

Figure 6: LTW versus In-out methods, $\alpha = 0.98$, webbase-2001 matrix.



(a) Percentage of time reduction with respect to the parallel Power method.

(b) Percentage of time reduction with respect to the parallel In-out Power method.

Figure 7: LTW versus Power method and In-out methods, DSM, $\alpha = 0.98$.

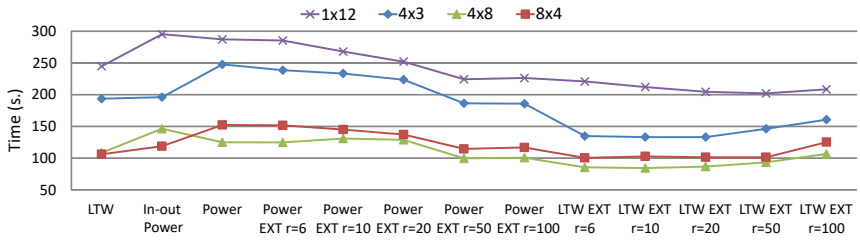


Figure 8: Comparison of algorithms, $\alpha = 0.98$, it-2004 matrix.

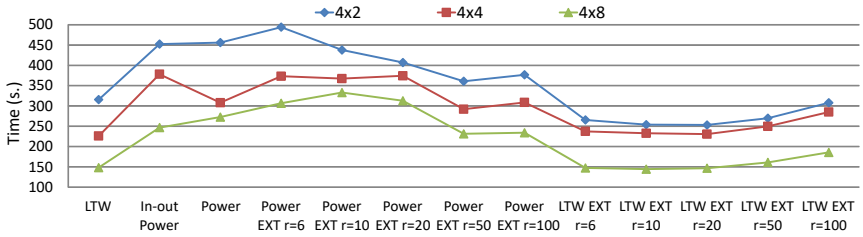


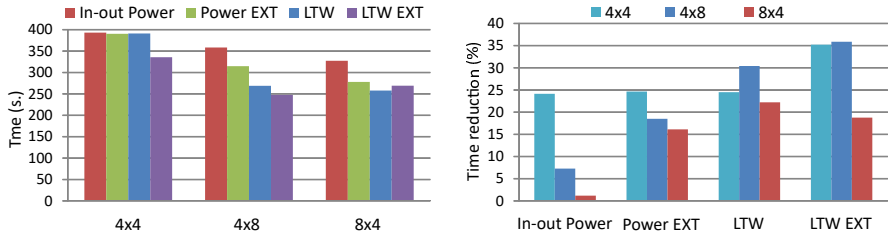
Figure 9: Comparison of algorithms, $\alpha = 0.98$, webbase-2001 matrix.

Graph	Iterations	Using *bvgraph* (s.)	Using $mCSR$ (s.)	ratio
it-2004	648	2217.49	387.91	5.72
webbase-2001	620	2240.18	623.98	3.59
uk-2006-10	654	3982.98	989.16	4.02
uk-2007-05	657	4920.34	1175.77	4.18

Table 7: Execution time (seconds) of the In-out Power method, *bvgraph* versus $mCSR$ format, SM (1×8), $\alpha = 0.99$.

of the OpenMP code proposed in [13] was analyzed using 8 cores and working with Web graphs compressed into *bvgraph* data structures. However, Table 7 shows that the use of the $mCSR$ storage format is much more efficient in terms of execution time.

Figure 6 analyzes the behaviour of the inner-outer algorithms compared with our LTW algorithm for several configurations of processes. As it can be expected, the inner-outer Power algorithm [13, Algorithm 2] performs better



(a) Parallel running times.

(b) Percentage of time reduction with respect to the parallel Power method.

Figure 10: LTW versus extrapolated Power and In-out methods, $\alpha = 0.98$, uk-2007-05 matrix.

than the basic inner-outer algorithm [13, Algorithm 1]. Nevertheless, the LTW algorithm outperforms these methods. Figure 7 compares the percentage of time reduction of the LTW algorithm with respect to both the Power and the inner-outer Power algorithms for the matrices of Table 1. The time saving of the LTW algorithm has been more significant for the webbase-2001 matrix obtaining a time reduction in relation to the parallel Power method of up to 46% and up to 40% with respect to the parallel inner-outer Power algorithm. For the other matrices, gains of up to 30% and 25% respectively, are achieved. Note that the density of the webbase-2001 matrix is the lowest, which yields lower computational work per iteration. Therefore, the communications between processes have more influence on the time needed to converge and hence, an algorithm with less communications can better improve the execution time.

Figures 8, 9 and 10 compare the LTW algorithm with the extrapolation methods [4]. These methods use successive iterates of the Power method to estimate the nonprincipal eigenvectors of the hyperlink matrix, and periodically subtracting these estimates from the current iterate of the Power method. In practice, the extrapolation only needs to be applied once to achieve maximum benefit. In this way, the parameter r in Figures 8 and 9 means that the extrapolation is applied once at the $r+2$ iteration of the Power method by means of the following expression $x^{(r+2)} = \frac{x^{(r+2)} - \alpha^r x^{(2)}}{1-\alpha^r}$. We want to point out that,

as it was shown in [5], the simple extrapolation ($r = 1$) is not effective and slows down the convergence of the Power method. It is due to the fact that the simple extrapolation assumes that α is the only eigenvalue of modulus α and this is inaccurate. Moreover, a good choice of the value of r for $\alpha = 0.85$ in the extrapolated Power method is $r = 6$, while for values of α close to 1, small values of r get poor results and it should be chosen greater than or equal to 50; see [4] and [5]. Concretely, for $\alpha = 0.98$, our results indicate that the best r is $r = 50$. On the other hand, we also analyze in these figures a combination of both the extrapolation and the LTW methods, that we have called LTW EXT algorithm. The idea is to start the LTW(q) iterations ($q > 1$) using an initial vector nearest to the solution. For this purpose, after the iterate $x^{(2)}$ is obtained by the Power method, r iterations of the LTW(1) algorithm are performed followed by an extrapolation. In this case, for $\alpha = 0.98$, we obtain good results using $4 \leq q \leq 6$ and $r = 20$. As it can be seen in these figures, the gain obtained by the best extrapolated methods in relation to the Power method is considerable, outperforming inner-outer Power methods. Furthermore, either the LTW algorithm or the extrapolated LTW algorithm remains the fastest of all of them.

ω	0.96	0.97	0.98	0.99	1	1.01	1.02	1.025
RTS(2)	324	322	321	319	318	316	315	318
RTS(4)	174	173	172	171	170	168	215	525

Table 8: Number of iterations of the RTS algorithm, SM (1×4), $\alpha = 0.98$, uk-2007-05 matrix.

To conclude the analysis of the experiments, we consider the relaxed version of the LTW algorithm, that is, the RTS algorithm. As it can be seen in Table 8, the RTS algorithm can reduce the number of iterations for the convergence of the PageRank problem when compared with its non-relaxed counterpart. For our problem models, good choices of the value of ω in the RTS algorithm are close to 1 but greater than 1. Nevertheless, the RTS algorithm only can save up to 3.5% in the time needed by the LTW to reach a residual of 10^{-6}. Concretely,

for the data of Table 8 the gain achieved with the RTS algorithm in relation to the LTW algorithm was of 3.35% for $q = 2$ and of 0.5% for $q = 4$. The choice of an optimal relaxed parameter for the RTS algorithm is problem dependent and taking into account that for a good ω both the RTS and LTW algorithms need similar number of iterations for convergence, it seems preferable to use the LTW algorithm instead of the RTS algorithm. Note that the RTS algorithm performs extra computation and this approach is advantageous only when the number of synchronizations (iterations) among processes can be significantly reduced.

8. Conclusion

In this work we have designed several parallel algorithms that use two-stage methods for solving the PageRank problem by means of its sparse linear system formulation. From a theoretical point of view, conditions on the proposed splittings are analyzed to guarantee the global convergence. The parallel code has been developed using MPI for data distribution among nodes and OpenMP to exploit loop level parallelism within each node. The formulation of the algorithm allows each process to calculate an approximation to the solution for a much smaller size problem than the original one. In addition, this technique aims to reduce the synchronization points at which a process must wait for information from other processes. In this sense, the nonzero element partitioning approach has achieved better load balancing among nodes than the row-wise distribution, since all processes terminate each global iteration at approximately the same time. The displayed numerical experiments show that the proposed parallel algorithms may be applied to large and sparse linear systems obtained from the PageRank problem, outperforming well-known methods such as the extrapolation Power methods [3] and the inner-outer techniques proposed in [13].

Acknowledgements

This research was supported by the Spanish Ministry of Economy and Competitiveness (MINECO) and the European Commission (FEDER funds) under Grant Number TIN2015-66972-C5-4-R.

References

[1] Page L, Brin S, Motwani R, Winograd T. The PageRank citation ranking: Bringing order to the Web. Technical Report, Stanford Digital Library Technologies Project; 1999.

[2] Wilkinson JH. The algebraic eigenvalue problem. Oxford: Oxford University Press; 1998.

[3] Kamvar SD, Haveliwala TH, Manning CD, Golub GH. Extrapolation methods for accelerating PageRank computations. In: Proceedings of the Twelfth International World Wide Web Conference, ACM Press; 2003, p. 261–270.

[4] Kamvar SD. Numerical algorithms for personalized search in self-organizing information networks. Princeton, New Jersey: Princeton University Press; 2010.

[5] Arnal J, Migallón H, Migallón V, Palomino JA, Penadés J. Parallel relaxed and extrapolated algorithms for computing PageRank. J Supercomput 2014;70:637–648.

[6] Kamvar SD, Haveliwala TH, Golub GH. Adaptive methods for the Computation of PageRank. Linear Algebra Appl 2004;386:51–65.

[7] Golub GH, Greif C. An Arnoldi-type algorithm for computing PageRank. BIT 2006;46(4):759–771.

[8] Wu G, Wei Y. An Arnoldi-Extrapolation algorithm for computing PageRank. J Comput Appl Math 2010;234:3196–3212.

[9] Wu G, Wei Y. A Power Arnoldi algorithm for computing PageRank. Numer Linear Algebra Appl 2007;14:521–546.

[10] Langville AN, Meyer CD. Google's Pagerank and Beyond: The Science of Search Engine Rankings. Princeton, New Jersey: Princeton University Press; 2006.

[11] Pua BY, Huanga TZ, Wena C. A preconditioned and extrapolation-accelerated GMRES method for PageRank. Appl Math Lett 2014;37:95–100.

[12] Gleich D, Zhukov L, Berkhin P. Fast parallel PageRank: A linear system approach. In: The Fourteenth International World Wide Web Conference. New York: ACM Press; 2005.

[13] Gleich D, Gray A, Greif C, Lau T. An inner-outer iteration for computing PageRank. SIAM J Sci Comput 2010;32(1):349–371.

[14] Xie YJ, Ma CF. A relaxed two-step splitting iteration method for computing PageRank. Comput Appl Math. 2016; DOI:10.1007/s40314-016-0338-4.

[15] Huang H, Ma CF. Parallel multisplitting iteration methods based on M-splitting for the PageRank problem. Appl Math Comput 2015;271:337–343.

[16] Gu C, Xie F, Zhang K. A two-step matrix splitting iteration for computing PageRank. J Comput Appl Math 2015;278:19–28.

[17] Boldi P, Vigna S. The webgraph framework I: Compression techniques. In: Proceedings of the Thirteenth International World Wide Web Conference. New York: ACM Press; 2004, p. 595-602.

[18] Migallón H, Migallón V, Penadés J. Parallel computation of PageRank using two-stage methods. In: Ivnyi P, Topping BHV, Vrady G, editors. Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Stirlingshire, UK: Civil-Comp Press; 2017, Paper 32, doi:10.4203/ccp.111.32.

[19] Bru R, Migallón V, Penadés J, Szyld DB. Parallel, synchronous and asynchronous two-stage multisplitting methods. Electron Trans Numer Anal 1995;3:24–38.

[20] Szyld DB, Jones MT. Two-stage and multisplitting methods for the parallel solution of linear systems. SIAM J Matrix Anal Appl 1992;13(2):671–679.

[21] Berman A, Plemmons RJ. Nonnegative Matrices in the Mathematical Sciences, 3rd ed. New York: Academic Press; 1979, reprinted by SIAM, Philadelphia, 1994.

[22] Frommer A, Szyld DB. H-splittings and two-stage iterative methods. Numer Math 1992;63:345–356.

[23] Lanzkron PJ, Rose DJ, Szyld DB. Convergence of nested classical iterative methods for linear systems. Numer Math 1991;58:685–702.

[24] Nichols NK. On the convergence of two-stage iterative processes for solving linear equations. SIAM J Numer Anal 1973;10(3):460–469.

[25] Dongarra J, Huss-Lederman S, Otto S, Snir M, Walkel D. MPI: The complete reference. 2nd ed. Cambridge, MA: The MIT Press; 1998.

[26] OpenMP official site; 2008. http://www.openmp.org.

[27] Byun JH, Lin R, Yelick KA, Demmel J. Autotuning sparse matrix-vector multiplication for multicore. Technical report UCB/EECS-2012-215, Electrical Engineering and Computer Sciences Department, UC Berkeley; 2012.

[28] Barrett R, Berry M, Chan TF, Demmel J, Donato J, Dongarra J, Eijkhout V, Pozo R, Romine C, Van der Vorst H. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. 2nd ed. Philadelphia, PA: SIAM Press; 1994.

[29] Kincaid DR, Young DM. A brief review of the ITPACK Project. J Comput Appl Math 1988;24(1-2):121–127.

[30] Vuduc, RW. Automatic Performance Tuning of Sparse Matrix Kernels. PhD thesis, UC Berkeley; 2003.

[31] Vázquez F, Fernández JJ, Garzón EM. A new approach for sparse matrix vector product on NVIDIA GPUs. Concurr Comput: Pract Expe 2011;23:815–826.

[32] Karakasis V, Goumas G, Koziris N. A comparative study of blocking storage methods for sparse matrices on multicore architectures. In: Proceedings of the International Conference on Computational Science and Engineering CSE '09; 2009, volume 1, p. 247–256.

[33] Laboratory for Web Algorithmics; 2002. http://law.di.unimi.it.

[34] Boldi P, Codenotti B, Santini M, Vigna S. Ubicrawler: A scalable fully distributed Web crawler. Softw Pract Expe 2004;34:711–726.

[35] Migallón H, Migallón V, Palomino JA, Penadés J. Parallelization strategies for computing PageRank. In: Topping BHV, Adam JM, Pallarés FJ, Bru R, Romero ML, editors. Proceedings of the Seventh International Conference on Engineering Computational Technology. Stirlingshire, United Kingdom: Civil-Comp Press; 2010, Paper 29, doi:10.4203/ccp.94.29.

[36] Bradley JT, de Jager DV, Knottenbelt WJ, Trifunović A. Hypergraph partitioning for faster parallel PageRank computation. In: Bravetti M, Kloul L, Zavattaro G, editors. Formal Techniques for Computer Systems and Business Processes. Lect Notes Comput Sc; 2005, volume 3670, p. 155–171.

[37] Trifunovic A, Knottenbelt WJ. Parkway 2.0: A parallel multilevel hypergraph partitioning tool. In: Aykanat C, Dayar T, Körpeoğlu I, editors. Computer and Information Science. Lect Notes Comput Sc; 2004, volume 3280, p. 789–800.