Hypergraph Partitioning for Faster Parallel PageRank Computation

Jeremy T. Bradley Douglas V. de Jager
William J. Knottenbelt Aleksandar Trifunović

Department of Computing, Imperial College London
180 Queen's Gate, London SW7 2BZ, United Kingdom.
{jb,dvd03,wjk,at701}@doc.ic.ac.uk

Abstract. The PageRank algorithm is used by search engines such as Google to order web pages. It uses an iterative numerical method to compute the maximal eigenvector of a transition matrix derived from the web's hyperlink structure and a user-centred model of web-surfing behaviour. As the web has expanded and as demand for user-tailored web page ordering metrics has grown, scalable parallel computation of PageRank has become a focus of considerable research effort.

In this paper, we seek a scalable problem decomposition for parallel PageRank computation, through the use of state-of-the-art hypergraph-based partitioning schemes. These have not been previously applied in this context. We consider both one and two-dimensional hypergraph decomposition models. Exploiting the recent availability of the Par*k*way 2.1 parallel hypergraph partitioner, we present empirical results on a gigabit PC cluster for three publicly available web graphs. Our results show that hypergraph-based partitioning substantially reduces communication volume over conventional partitioning schemes (by up to three orders of magnitude), while still maintaining computational load balance. They also show a halving of the per-iteration runtime cost when compared to the most effective alternative approach used to date.

1 Introduction

The PageRank metric is a widely-used hyperlink-based estimate of the relative importance of web pages [1]. The standard algorithm for determining PageRank uses power method iterations that converge to the maximal eigenvector of a transition matrix. This matrix is derived from a web graph that reflects the hyperlink structure of the web and a user-centred model of web-surfing behaviour.

The sheer size and high growth rate of the web necessitates a scalable parallel/distributed approach to PageRank computation. In turn, the scalability of such an approach demands detailed scrutiny of computation and communication overheads induced by problem decomposition over available processors. A poor decomposition results in excessive communication overhead and/or a poor computational load balance with correspondingly poor run times.

In addition to size considerations, web search engines are recognising the need to tailor search results to different classes of users (or individual users), and other contextual information [2]. This is achieved in practice by performing distinct PageRank computations (using distinct personalisation vectors) for each class of user or search context. Since these repeated calculations have the same structure (in terms of matrix sparsity pattern), it is often worthwhile investing considerable effort in finding a high-quality decomposition that can be reused for every PageRank calculation.

A promising state-of-the-art approach for producing high-quality decompositions (that has been used in many contexts, ranging from VLSI circuit layout to distributed database design), is hypergraph partitioning [3–5]. Hypergraphs are extensions of graph data structures, in which (hyper)edges connect arbitrary sets of vertices. Like graphs, hypergraphs can represent the structure of many sparse, irregular problems, and may be partitioned such that a cut metric is minimised subject to a load balancing constraint. However, hypergraph cut metrics provide a more expressive and accurate model than their graph counterparts. For example, in the decomposition of a sparse matrix for parallel matrix–vector multiplication, hypergraph models quantify communication volume exactly, whereas graph models can only provide an approximation [4].

This paper considers, for the first time, the application of hypergraph-based decomposition techniques to the parallel PageRank computation problem. We show how this problem can be mapped onto a hypergraph partitioning problem, for both one- and two-dimensional decompositions. The partitioning of hypergraphs of large scale has only recently become a practical proposition with the development of parallel hypergraph partitioning tools such as Par*k*way [6] and the forthcoming Zoltan implementation [7]. Exploiting the Par*k*way tool as part of a parallel PageRank computation pipeline, we present experimental results using a gigabit PC cluster on three public-domain web graphs, ranging in size from a university-domain to a national-level crawl. The results show a substantial reduction in per-iteration communication volume, yielding a runtime reduction of up to 70% over the most effective current alternative.

The remainder of this paper is organised as follows. Section 2 presents technical details of the PageRank algorithm. Section 3 describes the application of hypergraph partitioning to parallel PageRank computation. Section 4 discusses our results. Finally, Section 5 concludes and presents ideas for future work.

2 PageRank Algorithm

The PageRank computation for ranking hypertext-linked web pages was originally outlined by Page and Brin [1]. Later Kamvar *et al.* [8] presented a more rigorous formulation of PageRank and its computation. In fact, the latter description differs from the original; however, apart from the respective treatment of so-called *cul de sac* pages (web pages with no out-links, sometimes called dead-end pages), the difference is largely superficial [9]. A good discussion of the issues involved along with analysis of other variations in the PageRank algorithm can be found in Langville *et al.* [10]. We shall concern ourselves here with the Kamvar *et al.* formulation.

Two intuitive explanations are offered for PageRank [8]. The first presents PageRank as an analogue of citation theory: that is, an out-link from a web page w to a web page w' is an indication that w' may be "important" to the author of w. Many such links into w', especially from pages that are themselves "important", should raise the importance of w' relative to other web pages. More specifically, the importance that is propagated from w to w' should be proportional to the importance of w and inversely proportional to the number of out-links from w. This account of PageRank is still incomplete as it does not take into account any form of user *personalisation*, or how to deal with *cul de sac* pages.

The second conceptual model of PageRank is called the *random surfer* model. Consider a surfer who starts at a web page and picks one of the links on that page at random. On loading the next page, this process is repeated. If a *cul de sac* page is encountered, then the surfer chooses to visit a random page. During normal browsing, the user may also decide, with a fixed probability, not to choose a link from the current page, but instead to jump at random to another page. In the latter case, to support both unbiased and personalised surfing behaviour, the model allows for the specification of a probability distribution of target pages.

The PageRank of a page is considered to be the (steady-state) probability that the surfer is visiting a particular page after a large number of click-throughs. Calculating the steady-state probability vector corresponds to finding a maximal eigenvalue of the modified web-graph transition matrix. As shown in Section 2 below, this can be done via an iterative numerical method based on sparse matrix–vector multiply operations.

Random Surfer Model In the random surfer model, the web is represented by a graph $G = (V, E)$, with web pages as the vertices, V, and the links between web pages as the edges, E. If a link exists from page u to page v then $(u \rightarrow v) \in E$.

To represent the following of hyperlinks, we construct a transition matrix $\mathbf{P}$ from the web graph, setting:

$$p_{ij} = \begin{cases} \frac{1}{\deg(u_i)} & : \text{if } (u_i \rightarrow u_j) \in E \\ 0 & : \text{otherwise} \end{cases}$$

where $\deg(u)$ is the out-degree of vertex u, i.e. the number of outbound links from page u. From this definition, we see that if a page has no out-links, then this corresponds to a zero row in the matrix $\mathbf{P}$.

To represent the surfer's jumping from *cul de sac* pages, we construct a second matrix $\mathbf{D} = \mathbf{d}\mathbf{p}^T$, where $\mathbf{d}$ and $\mathbf{p}$ are both column vectors and:

$$d_i = \begin{cases} 1 : \text{if } \deg(u_i) = 0 \\ 0 : \text{otherwise} \end{cases}$$

and $\mathbf{p}$ is the personalisation vector representing the probability distribution of destination pages when a random jump is made. Typically, this distribution is taken to be uniform, i.e. $p_i = 1/n$ for an n-page graph $(1 \leq i \leq n)$. However, it need not be as many distinct personalisation vectors may be used to represent different classes of user with different web browsing patterns. This flexibility comes at a cost, though, as each distinct personalisation vector requires an additional PageRank calculation.

Putting together the surfer's following of hyperlinks and his/her random jumping from *cul de sac* pages yields the stochastic matrix $\mathbf{P}' = \mathbf{P} + \mathbf{D}$, where $\mathbf{P}'$ is a transition matrix of a discrete-time Markov chain (DTMC).

To represent the surfer's decision not to follow any of the current page links, but to instead jump to a random web page, we construct a *teleportation* matrix $\mathbf{E}$, where $e_{ij} = p_j$ for all i, i.e. this random jump is also dictated by the personalisation vector.

Incorporating this matrix into the model gives:

$$\mathbf{A} = c\mathbf{P}' + (1-c)\mathbf{E} \tag{1}$$

where $0 < c < 1$, and c represents the probability that the user chooses to follow one of the links on the current page, i.e. there is a probability of $(1-c)$ that the surfer randomly jumps to another page instead of following links on the current page.

This definition of $\mathbf{A}$ avoids two potential problems. The first is that $\mathbf{P}'$, although a valid DTMC transition matrix, is not necessarily irreducible (i.e. it might have more than one strongly connected subset of states) and aperiodic. Taken together, these are a sufficient condition for the existence of a unique steady-state distribution. Now, provided $p_i > 0$ for all $1 \leq i \leq n$, irreducibility and aperiodicity are trivially guaranteed.

The second problem relates to the rate of convergence of power method iterations used to compute the steady-state distribution. This rate depends on the reciprocal of the modulus of the subdominant eigenvalue (λ_2). For a general $\mathbf{P}'$, $|\lambda_2|$ may be very close to 1, resulting in a very poor rate of convergence. However, it has been shown in [11] that in the case of matrix $\mathbf{A}$, $|\lambda_2| \leq c$, thus guaranteeing a good rate of convergence for the widely taken value of $c = 0.85$.

Given the matrix $\mathbf{A}$, we can now define the unique PageRank vector, π, to be the steady-state vector or the maximal eigenvector that satisfies:

$$\pi\mathbf{A} = \pi \tag{2}$$

Power Method Solution Having constructed $\mathbf{A}$ we might naïvely attempt to find the PageRank vector of Eq. (2) by using a direct power method approach:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}\mathbf{A} \tag{3}$$

where $\mathbf{x}^{(k)}$ is the kth iterate towards the PageRank vector, π. However looking at the current size of the web and its rate of growth since 1997 (currently 8 billion indexed pages [12]), it is clear that this is not a practical approach for realistic web graphs. The reason for this is that $\mathbf{A}$ is a (completely) dense matrix. Accordingly, the PageRank algorithm, as cited in for instance Kamvar *et al.* [13], reduces Eq. (3) to a series of sparse vector–matrix operations on the original $\mathbf{P}$ matrix.

In particular, transforming Eq. (3) gives:

$$\begin{aligned}\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)}\mathbf{A}\\ &= c\mathbf{x}^{(k)}\mathbf{P}' + (1-c)\mathbf{x}^{(k)}\mathbf{E}\\ &= c\mathbf{x}^{(k)}\mathbf{P} + c\mathbf{x}^{(k)}\mathbf{D} + (1-c)\mathbf{x}^{(k)}(\mathbf{1}\mathbf{p}^T)\end{aligned} \tag{4}$$

Now $\mathbf{x}^{(k)}\mathbf{D} = (\|\mathbf{x}^{(k)}\|_1 - \|\mathbf{x}^{(k)}\mathbf{P}\|_1)\mathbf{p}^T$, where $\|\mathbf{a}\|_1 = \sum_i |a_i|$ is the 1-norm of $\mathbf{a}$ and further $\|\mathbf{a}\|_1 = \mathbf{1}^T\mathbf{a}$ if $a_i \geq 0$ for all i. It can be shown inductively that $\|\mathbf{x}^{(k)}\|_1 = 1$ for all k, so:

$$\begin{aligned}\mathbf{x}^{(k+1)} &= c\mathbf{x}^{(k)}\mathbf{P} + c(1 - \|\mathbf{x}^{(k)}\mathbf{P}\|_1)\mathbf{p}^T + (1-c)(\mathbf{x}^{(k)}\mathbf{1})\mathbf{p}^T\\ &= c\mathbf{x}^{(k)}\mathbf{P} + (1 - c\|\mathbf{x}^{(k)}\mathbf{P}\|_1)\mathbf{p}^T\end{aligned} \tag{5}$$

This leads to the algorithm shown in Fig. 1. When distributing this algorithm, it is important to distribute the sparse matrix–vector calculation of $\mathbf{x}^{(k)}\mathbf{P}$ in such a way so as to balance computational load as evenly as possible across the processors and minimise communication overhead between processors. This latter optimisation is where we introduce hypergraph partitioning for $\mathbf{P}$.

Later, in Section 4, we refine the coarse notion of communication overhead to distinguish between number of messages sent, total communication volume (in terms of number of floating point elements sent), as well as maximum number of messages sent by a processor.

1. $\mathbf{x}^{(0)} := \mathbf{p}^T$
2. $\mathbf{y} := c\mathbf{x}^{(k)}\mathbf{P}$
3. $\omega := \|\mathbf{x}^{(k)}\|_1 - \|\mathbf{y}\|_1$
4. $\mathbf{x}^{(k+1)} := \mathbf{y} + \omega\mathbf{p}^T$
5. Repeat from 2. until $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|_1 < \epsilon$

Fig. 1. Pseudocode description of the PageRank algorithm.

3 Parallel PageRank Computation

We consider the parallel formulation of the PageRank algorithm from Section 2 for which the kernel operation is parallel sparse matrix–vector multiplication. Note that, although our discussion is presented in the context of power method solution, there is nothing to prevent the application of our technique to other iterative linear system solvers with a sparse matrix–vector multiplication kernel, such as the Krylov subspace methods proposed in [14]. Furthermore, our approach does not preclude the application of power method acceleration techniques, for example those proposed in [8].

Efficient Parallel Sparse Matrix–Vector Multiplication Let $\mathbf{Ax} = \mathbf{b}$ be the sparse matrix–vector product to be computed in parallel on p distributed processors that are connected by a network. The general form of a parallel algorithm for sparse matrix–vector multiplication with an arbitrary non-overlapping distribution of the matrix and the vectors across the processors is given in [5]:

1. Each processor sends its components x_j to those processors that possess a non-zero a_{ij} in column j.
2. Each processor computes the products $a_{ij}x_j$ for its non-zeros a_{ij} and adds the results for the same row index i. This yields a set of contributions b_{is}, where s is the processor identifier $0 \le s < p$.
3. Each processor sends its non-zero contributions b_{is} to the processor that is assigned vector element b_i.
4. Each processor adds the contributions received for its components b_i, giving $b_i = \sum_{s=0}^{p-1} b_{is}$.

Efficient parallel sparse matrix–vector multiplication requires intelligent *a priori* partitioning of the sparse matrix non-zeros across the processors. This ensures that interprocessor communication during stages 1 and 3 is minimised and computational load balance is achieved across the processors. We note that the computational requirement of step 2 dominates that of step 4. Henceforth, we assume that the computational load of the entire algorithm is represented by step 2.

Recently, a number of hypergraph-based models for parallel sparse matrix–vector multiplication that correctly model total communication volume and per-processor computational load have been proposed [4, 15, 16, 5]. These have addressed the shortcomings implicit in traditional graph models [17]. In [4], a hypergraph-based model for 1-dimensional decomposition of the sparse matrix is proposed. A 1-dimensional decomposition implies that processors either store entire rows or entire columns of the matrix. Note that, in the case of row-wise decomposition, this has the effect of making the communication step 3 in the parallel sparse matrix–vector multiplication pipeline redundant; in the case of column-wise decomposition, step 1 is redundant. The hypergraph-based models in [15, 16, 5] are 2-dimensional, which means to say that they model a general distribution of matrix non-zeros to processors (not necessarily assigning entire rows or columns of the matrix to processors). Although here both steps 1 and 3 may incur communication overhead, the overall communication volume should be at least as low as that of the 1-dimensional decomposition (at least for optimal partitions, since the 1-dimensional decomposition is a special case of the 2-dimensional decomposition).

In previous work on efficient parallel PageRank implementation, only naïve 1-dimensional matrix decompositions have been considered. In [14], the authors reject traditional graph partitioning models as a plausible approach, on account of the apparent power-law distribution of the number of non-zeros in the rows of web graph transition matrices. Instead, they use a relatively simple load balancing scheme that assigns consecutive rows of the matrix to each processor. Our work here demonstrates that this power-law distribution does not appear to be a significant obstacle in the context of a hypergraph-based approach.

In this paper, we consider both 1-dimensional decomposition, based on the hypergraph model presented in [4], and 2-dimensional decomposition, based on the models presented in [15, 5]. Since the output vector $\mathbf{b}$ of the parallel sparse matrix–vector product is reused as the input vector $\mathbf{x}$ in the subsequent iteration, we note that the processor that is assigned the vector component b_i should also be assigned the vector component x_i (resulting in a *symmetric* decomposition of the vector elements).

Description of the Hypergraph Models A hypergraph is a set system $(V, \mathcal{E})$ on a set V, here denoted $H(V, \mathcal{E})$, such that $\mathcal{E} \subset \mathcal{P}(V)$, where $\mathcal{P}(V)$ is the power set of V. The elements of the set V are called the *vertices* of the hypergraph and $\mathcal{E}$ the set of *hyperedges*, where each hyperedge $e \in \mathcal{E}$ is a subset of the set V. When $\mathcal{E} \subset V^{(2)}$, each hyperedge has cardinality two and the resulting set system is known as a *graph*. A hypergraph $H(V, \mathcal{E})$ is said to be hyperedge-weighted if each hyperedge $e \in \mathcal{E}$ has an associated integer weight. Correspondingly, in a vertex-weighted hypergraph $H(V, \mathcal{E})$, each vertex $v \in V$ has an integer weight.

Given a hypergraph $H(V, \mathcal{E})$, with $V = \{v_1, \ldots, v_n\}$ and $\mathcal{E} = \{e_1, \ldots, e_n\}$, the corresponding *incidence matrix* $\mathbf{A} = (a_{ij})$ is the $n \times n$ matrix with entries

$$a_{ij} = \begin{cases} 1 \text{ if } v_i \in e_j \\ 0 \text{ otherwise} \end{cases} \tag{6}$$

A k-way *partition* Π $(k > 1)$ of the set V is a finite collection $\Pi = \{P_1, \ldots, P_k\}$, of subsets of V (or parts), such that $P_i \cap P_j = \emptyset$ for all $1 \le i < j \le k$ and $\bigcup_{i=1}^{k} P_i = V$. A hyperedge is said to be cut by a partition if it spans (i.e. has a vertex in) at least two parts of a partition. The goal of the hypergraph partitioning problem is to find a k-way partition $\Pi = \{P_1, \ldots, P_k\}$, with corresponding part weights W_i, $1 \leq i \leq k$, such that an objective function $f_o : (\Pi, \mathcal{E}) \rightarrow \mathbb{Z}$ is optimised, while a balance constraint over the part weights is maintained. That is, for some ϵ ($0 < \epsilon \ll 1$):

$$W_i < (1+\epsilon)W_{avg} \tag{7}$$

for all $1 \leq i \leq k$. The part weights W_i are computed as the sum of the constituent vertex weights. Here we consider the objective function known as the $k-1$ metric, shown in Eq. (8), where λ_i represents the number of parts spanned by hyperedge e_i and $w(e_i)$ is the weight of hyperedge e_i.

$$f_o(\Pi, \mathcal{E}) = \sum_{i=1}^{n} (\lambda_i - 1)w(e_i) \tag{8}$$

As the hypergraph partitioning problem is NP-hard [18], in practice a good sub-optimal partition is sought in low-order polynomial time using heuristic multilevel algorithms.

1-Dimensional Sparse Matrix Decomposition Without loss of generality, we describe the hypergraph model for 1-dimensional row-wise sparse matrix decomposition, i.e. where all non-zeros in a row of the matrix are allocated to the same processor. A similar column-wise model follows from considering the allocation of all non-zeros in a column of the matrix to the same processor. These 1-dimensional hypergraph-based models were first proposed in [4].

The hypergraph model $H(V, \mathcal{E})$ for the decomposition of a sparse matrix $\mathbf{A}$ is constructed as follows. The rows of the matrix $\mathbf{A}$ form the set of vertices V in the hypergraph $H(V, \mathcal{E})$ and the columns form the set of hyperedges $\mathcal{E}$. That is, if $a_{ij} \neq 0$, then hyperedge $e_j \in \mathcal{E}$, defined by column j of the matrix $\mathbf{A}$, contains vertex $v_i \in V$. The weight of vertex $v_i \in V$ is given by the number of non-zero elements in row i of the matrix $\mathbf{A}$, representing the computational load induced by assigning row i to a processor. The weights of each hyperedge are set to unity.

The allocation of the rows of the matrix $\mathbf{A}$ to p processors for parallel sparse matrix–vector multiplication corresponds to a p-way partition Π of the above hypergraph $H(V, \mathcal{E})$. Ignoring the negligible impact of stage 4, as mentioned earlier, the computational load on each processor i is given by the number of scalar multiplications performed on that processor during stage 2 of the general parallel sparse matrix–vector multiplication pipeline. This quantity is given by the number of non-zeros of the matrix $\mathbf{A}$ allocated to that processor, which is in turn given by the weight of part P_i.

The vector elements x_i and b_i are allocated to the processor that is allocated row i of the matrix $\mathbf{A}$. There remains one further condition that the hypergraph model must satisfy to ensure that the k-1 metric on partition Π exactly represents the total communication volume incurred during a single parallel sparse matrix–vector multiplication (in this case stage 1 only). We require that for all $1 \leq i \leq n$, $v_i \in e_i$ holds. If this is not the case for some $1 \leq i' \leq n$, then we add $v_{i'}$ to hyperedge $e_{i'}$. The weight of $v_{i'}$ is not modified.

Thus, finding a partition that minimises the k-1 metric over the hypergraph while maintaining the balance constraint in Eq. (7) directly corresponds to minimising communication volume during parallel sparse matrix–vector multiplication while maintaining a computational load balance.

2-Dimensional Sparse Matrix Decomposition The 2-dimensional sparse matrix decomposition takes a more general approach, no longer imposing the restriction of allocating entire rows (or columns) of the matrix $\mathbf{A}$ to the same processor, as in 1-dimensional decomposition. Instead, a general distribution of matrix non-zeros to processors is considered. This may introduce additional communication operations during stage 3 in the general parallel sparse matrix–vector multiplication pipeline, but the aim is to reduce the overall communication volume. Here we describe the hypergraph model $H(V, \mathcal{E})$ introduced in [15, 5]. Each non-zero $a_{ij} \neq 0$ is modelled by a vertex $v \in V$ so that a p-way partition Π of the hypergraph $H(V, \mathcal{E})$ will correspond to an assignment of matrix non-zeros across p processors.

In order to define the hyperedges of the hypergraph model, consider the cause of communication between processors in stages 1 and 3 of the parallel sparse matrix–vector multiplication pipeline. In stage 1, the processor with non-zero a_{ij} requires vector element x_j for computation during stage 2. This results in a communication of x_j to the processor assigned a_{ij} if x_j is assigned to a different processor to a_{ij}. The dependence between non-zeros in column j of matrix $\mathbf{A}$ and vector element x_j can be modelled by a hyperedge, whose constituent vertices are the non-zeros of column j of the matrix $\mathbf{A}$. So that the communication volume associated with communicating vector element x_j is given by $\lambda_j - 1$, where λ_j denotes the number of parts spanned by the column j hyperedge, we require that the column j hyperedge contains the vertex corresponding to non-zero a_{jj}. If a_{jj} is zero in the matrix $\mathbf{A}$, we can add a "dummy" vertex with zero weight corresponding to a_{jj}. The fact that this vertex has weight zero means that its allocation to a processor will have no bearing on the processor's computational load, while the exact communication volume during stage 1 is modelled correctly.

In stage 3, the processor assigned vector element b_i requires the value of the inner product of row i of the matrix $\mathbf{A}$ with the vector $\mathbf{x}$. Thus, a communication between processors is induced if matrix non-zero a_{ij} is assigned to a different processor from vector entry b_i. The dependence between non-zeros in row i of matrix $\mathbf{A}$ and vector element b_i can be modelled by a hyperedge, whose constituent vertices are the non-zeros of row i of the matrix $\mathbf{A}$. This is analogous to modelling the communication of stage 1 with column hyperedges and likewise, "dummy" vertices corresponding to a_{ii} are added to row hyperedge i if the value of a_{ii} in matrix $\mathbf{A}$ is zero.

The hypergraph model $H(V, \mathcal{E})$ is then partitioned into p parts such that the k-1 metric is minimised, subject to the balance constraint of Eq. (7) (thus maintaining computational load balance during stage 2). In our implementation, this is done using a parallel multilevel partitioning algorithm [19, 6].

Note that, except for restricting vector elements x_i and b_i to the same processor, we have not explicitly allocated vector entries to processors. The overall communication volume during the parallel sparse matrix–vector multiplication will be correctly modelled by the 2-dimensional hypergraph model, provided that we allocate the vector elements to processors in the following fashion. For the vector element with index i:

1. If both the row i hyperedge and the column i hyperedge are not cut, then assign vector elements x_i and b_i to the processor assigned vertices from row i and column i hyperedges.
2. If the row i hyperedge is cut and the column i hyperedge is not cut, then assign vector elements x_i and b_i to the processor assigned vertices from column i hyperedge.
3. If the row i hyperedge is not cut and the column i hyperedge is cut, then assign vector elements x_i and b_i to the processor assigned vertices from row i hyperedge.
4. If both the row i hyperedge and the column i hyperedge are cut, then let R_i denote the set of processors that contain row i hyperedge elements and let C_i denote the set of processors that contain column i hyperedge elements. Since either $a_{ii} \neq 0$ or there exists a "dummy" vertex in the row i and column i hyperedges corresponding to a_{ii}, the set $T_i = R_i \cap C_i$ is non-empty and vector elements x_i and b_i may be assigned to any of the processors in T_i.

With the additional freedom in the assignment of vector elements to processors given by case 4 above, it may be possible to further decrease the *maximum* number of messages sent or received by an individual processor while keeping the overall communication volume constant. In our implementation of stage 4 above, the vector elements x_i and b_i are allocated to the first part in T_i encountered during traversal of matrix $\mathbf{A}$.

4 Experimental Results

In this section, we apply four decomposition strategies to calculate PageRanks for three publicly available web graphs. Each web graph was generated from a crawl of a particular domain or combination of domains; we represent them by a sparse matrix $\mathbf{A}$ with non-zero a_{ij} whenever there exists a link from page i

WebGraph	#rows	#columns	#non-zeros
Stanford	281 903	281 903	2 594 228
Stanford_Berkeley	683 446	683 446	8 262 087
India-2004	1 382 908	1 382 908	16 917 053

Tab. 1. Characteristics of the test hypergraphs

to page j. The Stanford and Stanford_Berkeley web graphs were obtained from the University of Florida Sparse Matrix Collection [20] and represent lexically ordered crawls of the Stanford and combined Stanford/Berkeley domains respectively. The India-2004 web graph represents a breadth-first crawl of the `.in` domain conducted in 2004, obtained from the UbiCrawler public data set [21]. The main characteristics of the corresponding matrices are given in Table 1.

The two hypergraph decomposition methods of Section 3 were tested against two naïve load balancing methods. In the cyclic row-striping matrix decomposition, the non-zeros of the matrix $\mathbf{A}$ in row with index i are assigned to the processor $i \bmod p$. Vector elements x_i and b_i are also allocated to processor $i \bmod p$. This ensures that each processor is allocated the same number (± 1) of rows of matrix $\mathbf{A}$ and vector elements of $\mathbf{x}$ and $\mathbf{b}$. However, this scheme does not take into account the distribution of the non-zeros within the rows.

The load balancing scheme presented in [14], hereafter referred to as the GleZhu scheme, attempts to balance the number of non-zeros across the processors, while assigning consecutive rows of the matrix $\mathbf{A}$ to each processor. A threshold value $\tau_p = (w_n n + w_\eta \eta)/p$ is computed, where n is the number of rows and η the number of non–zeros in the matrix. The parameters w_n and w_η were both set to unity in [14]. Starting with row index zero and $i = 0$, the load–balancing algorithm then assigns consecutive rows of matrix $\mathbf{A}$ and consecutive elements of vectors $\mathbf{x}$ and $\mathbf{b}$ to each processor i, maintaining the value of $\tau_i = w_n n_i + w_\eta \eta_i$, where n_i is the number of rows and η_i the number of non–zeros assigned thus far to processor i. When τ_i exceeds τ_p, the algorithm begins to assign subsequent rows to processor $i + 1$.

Experimental Setup Our parallel PageRank computation pipeline is shown in Fig. 2. Taking the web graph matrix $\mathbf{A}$ as input, a decomposition of this matrix across p processors is performed using either one of the hypergraph partitioning-based models (i.e. 1D or 2D) or one of the load balancing row-wise decomposition methods (i.e. cyclic or GleZhu).

The hypergraph partitioning-based schemes compute a p-way partition of the hypergraph representation of the sparse web matrix using the parallel hypergraph partitioning tool Par$k$way2.1 [6]. In our experiments, we have used a 5% balance constraint for hypergraph partitioning, meaning that the weight of each part in the partition of the hypergraph must not exceed the average part weight by more than 5% ($\epsilon = 0.05$ in Eq. (7)). The computed hypergraph partition is

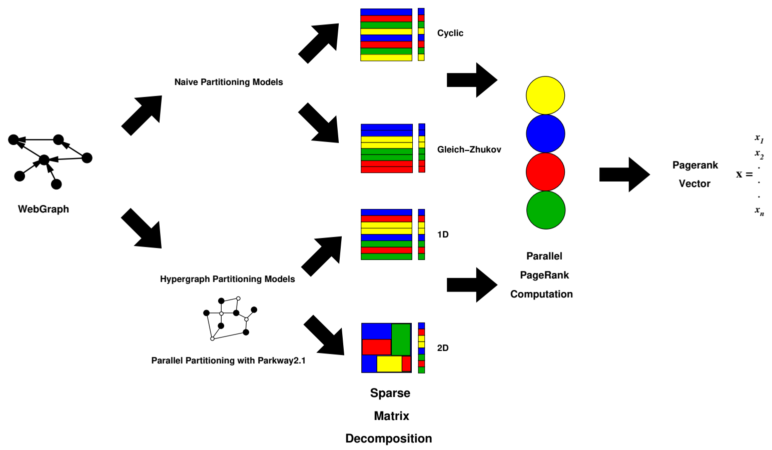

Fig. 2. Parallel PageRank Calculation Pipeline

then used to allocate the rows (in the case of 1D partitioning) or the non-zeros (in the case of 2D partitioning) of the web matrix to the processors.

Finally, the algorithm described in Section 2 is used to compute the PageRank vector for the matrix, with all matrix–vector and vector operations performed in parallel. The criterion of convergence for the PageRank calculation was taken to be 10^{-8} and convergence was computed using the L_1 norm.

The architecture used in all the experiments consisted of a Beowulf Linux Cluster with 8 dual processor nodes. Each node has two Intel Pentium 4 3.0GHz processors and 2GB of RAM. The nodes are connected by a gigabit Ethernet network. The algorithms were implemented in C++ using the Message Passing Interface (MPI) standard.

Results For each matrix decomposition method, we observed the following measures of communication cost during each parallel PageRank iteration: total communication volume (the total volume of all messages sent); number of messages sent; the maximum total communication volume of messages sent and received by a single processor during stage 1, in the case of row-wise decomposition, and the maximum total communication volume of messages sent and received by a single processor during stages 1 and 3, in the case of 2D decomposition.

The purely load balancing matrix decomposition approaches do not attempt to minimise the metrics above. The 1D and 2D hypergraph-based methods aim to minimise the overall communication volume. In row-wise decomposition methods, the number of messages sent during parallel sparse matrix–vector multipli-

$p = 4$	**Cyclic**	**GleZhu**	**1D hypergraph**	**2D hypergraph**
decomposition time(s)	Neg.	Neg.	15.2	1 181
#iterations	83	83	85	88
per iteration time(s)	0.2153	0.1681	0.0699	0.0762
$\mathbf{Ax = b}$ **time(s)**	0.2028	0.1621	0.0583	0.0657
$\mathbf{Ax = b}$ **comp. time(s)**	0.0607	0.0390	0.0551	0.0599
$\mathbf{Ax = b}$ **comm. time(s)**	0.1427	0.1237	0.0035	0.0058
#messages	12	12	12	19
max non-zeros per proc.	614 346	583 653	607 030	601 362
max vector elems per proc.	70 476	73 611	90 601	87 253
max per proc. comm. vol.	304 442	267 683	12 344	1 318
total comm. vol.	601 964	530 420	13 849	1 399
$p = 8$	**Cyclic**	**GleZhu**	**1D hypergraph**	**2D hypergraph**
decomposition time(s)	Neg.	Neg.	13.2	1 061
#iterations	79	79	83	86
per iteration time(s)	0.1854	0.1473	0.0443	0.0465
$\mathbf{Ax = b}$ **time(s)**	0.1716	0.1415	0.0318	0.0365
$\mathbf{Ax = b}$ **comp. time(s)**	0.0425	0.0169	0.0269	0.0309
$\mathbf{Ax = b}$ **comm. time(s)**	0.1299	0.1253	0.0055	0.0056
#messages	56	56	44	64
max non-zeros per proc.	326 891	297 854	303 515	299 503
max vector elems per proc.	35 238	38 962	49 443	55 398
max per proc. comm. vol.	255 053	231 233	31 564	1 660
total comm. vol.	989 071	894 098	34 221	2 285
$p = 16$	**Cyclic**	**GleZhu**	**1D hypergraph**	**2D hypergraph**
decomposition time(s)	Neg.	Neg.	18.3	543.1
#iterations	75	76	79	81
per iteration time(s)	0.1810	0.1446	0.0515	0.0513
$\mathbf{Ax = b}$ **time(s)**	0.1614	0.1377	0.0347	0.0353
$\mathbf{Ax = b}$ **comp. time(s)**	0.0532	0.0182	0.0242	0.0277
$\mathbf{Ax = b}$ **comm. time(s)**	0.1094	0.1203	0.0116	0.0076
#messages	240	240	147	207
max non-zeros per proc.	192 857	155 898	151 757	151 236
max vector elems per proc.	17 619	21 208	31 215	28 221
max per proc. comm. vol.	186 331	173 525	39 820	2 214
total comm. vol.	1 364 285	1 325 808	74 137	4 307

Tab. 2. Stanford Web graph Results

cation is at most $p(p-1)$. In the 2D method, the number of messages is at most $2p(p-1)$.

Tables 2, 3 and 4 present results of our experiments on the Stanford, Stanford_Berkeley and india-2004 web graphs, respectively. The following statistics are also recorded, for the combination of different web graph models being run on 4, 8 and 16 processor clusters using the 4 distinct partitioning algorithms: decomposition time (time taken to prepare the partition for each of the different partitioning algorithms); number of iterations (number of iterations to convergence of the distributed PageRank algorithm); per iteration times (average time for a single PageRank iteration); $\mathbf{Ax} = \mathbf{b}$ time (average time to perform a single $\mathbf{Ax} = \mathbf{b}$ iteration); $\mathbf{Ax} = \mathbf{b}$ comp. time (time taken to complete the local computation of an $\mathbf{Ax} = \mathbf{b}$ iteration); $\mathbf{Ax} = \mathbf{b}$ comm. time (time taken to complete the interprocessor communication of an $\mathbf{Ax} = \mathbf{b}$ iteration); Max non-zeros per proc. (maximum number of non-zeros allocated per processor); Max vector elems per proc. (maximum number of vector elements allocated per processor); Max per proc. comm vol. (maximum communication volume sent and received by a processor); Total comm. vol. (total communication volume of number of floating point elements sent in a single PageRank iteration).

Note that, due to numerical errors (truncation and roundoff), the number of iterations is not constant across the different methods. We observe that the application of hypergraph partitioning attracts a significantly lower overall communication overhead. 2D partitioning is the most effective at reducing overall communication volume, although this does not always translate into a lower

$p = 4$	**Cyclic**	**GleZhu**	**1D hypergraph**	**2D hypergraph**
decomposition time(s)	Neg.	Neg.	22.9	5 169
#iterations	84	87	89	89
per iteration time(s)	0.4596	0.0618	0.0353	0.0377
$\mathbf{Ax} = \mathbf{b}$ **time(s)**	0.4341	0.0527	0.0253	0.0264
$\mathbf{Ax} = \mathbf{b}$ **comp. time(s)**	0.0632	0.0237	0.0239	0.0244
$\mathbf{Ax} = \mathbf{b}$ **comm. time(s)**	0.3714	0.0293	0.0018	0.0019
#messages	12	12	12	20
max non-zeros per proc.	1 977 527	1 906 240	1 990 554	1 989 151
max vector elems per proc.	170 862	188 568	204 129	243 758
max per proc. comm. vol.	810 530	112 101	6 432	2 023
total comm. vol.	1 605 286	165 765	6 648	2 081
$p = 8$	**Cyclic**	**GleZhu**	**1D hypergraph**	**2D hypergraph**
decomposition time(s)	Neg.	Neg.	18.4	3 304
#iterations	80	85	85	84
per iteration time(s)	0.4616	0.0458	0.0285	0.0246
$\mathbf{Ax} = \mathbf{b}$ **time(s)**	0.4376	0.0395	0.0202	0.0167
$\mathbf{Ax} = \mathbf{b}$ **comp. time(s)**	0.0774	0.0123	0.0136	0.0130
$\mathbf{Ax} = \mathbf{b}$ **comm. time(s)**	0.3578	0.0276	0.0071	0.0038
#messages	56	56	42	62
max non-zeros per proc.	1 063 001	961 340	994 257	994 592
max vector elems per proc.	85 431	115 805	131 713	142 253
max per proc. comm. vol.	727 768	129 977	35 117	2 620
total comm. vol.	2 744 682	269 095	45 132	3 479
$p = 16$	**Cyclic**	**GleZhu**	**1D hypergraph**	**2D hypergraph**
decomposition time(s)	Neg.	Neg.	18.8	1 842
#iterations	76	85	85	83
per iteration time(s)	0.5955	0.0518	0.0351	0.0238
$\mathbf{Ax} = \mathbf{b}$ **time(s)**	0.5549	0.0443	0.0271	0.0150
$\mathbf{Ax} = \mathbf{b}$ **comp. time(s)**	0.1435	0.0110	0.0101	0.0102
$\mathbf{Ax} = \mathbf{b}$ **comm. time(s)**	0.4132	0.0340	0.0169	0.0048
#messages	240	178	129	165
max non-zeros per proc.	627 253	510 616	497 659	497 055
max vector elems per proc.	42 716	73 665	78 873	69 754
max per proc. comm. vol.	548 922	120 589	80 112	3 242
total comm. vol.	4 002 962	478 162	147 590	7 302

Tab. 3. Stanford Berkeley Web graph results

PageRank per-iteration time, on account of the higher number of messages sent, and the relatively high message start-up cost on our gigabit PC cluster.

Fig. 3 displays the total per-iteration communication volume for each partitioning algorithm. It shows that the GleZhu technique has a lower communication overhead than the naïve cyclic partitioning, as might be expected. We also see that, when compared to the GleZhu method, hypergraph partitioning reduces communication volume by an order of magnitude for 1D hypergraph partitioning and by 2 orders of magnitude for 2D hypergraph partitioning.

Fig. 4 shows the overall PageRank iteration time for GleZhu, 1D and 2D hypergraph partitions of the Stanford_Berkeley web matrix on the 16-processor cluster. The *computation* label refers to the time taken to compute a single $\mathbf{Ax} = \mathbf{b}$ iteration. The *communication* label represents the time taken in communication when performing a single $\mathbf{Ax} = \mathbf{b}$ iteration. The results for the cyclic technique are not shown as they are orders of magnitude larger and our main interest here is in comparing the GleZhu method (as the best currently used alternative) with the hypergraph versions. We see that the overall PageRank iteration time is dictated by the communication overhead incurred in performing the distributed $\mathbf{Ax} = \mathbf{b}$ calculation. As might be expected, the computation element and the residual of the PageRank computation (those calculations not involving the distributed matrix–vector multiplication) of the algorithm contribute an (approximately) fixed cost to the overall iteration time. We observe that 1D and 2D hypergraph partitioning successfully reduce the communication overhead by

$p = 4$	Cyclic	GleZhu	1D hypergraph	2D hypergraph
decomposition time(s)	Neg.	Neg.	557.5	13 480
#iterations	81	84	84	85
per iteration time(s)	0.7577	0.1142	0.0762	0.0781
Ax = b time(s)	0.7094	0.0972	0.0537	0.0528
Ax = b comp. time(s)	0.1243	0.0501	0.0526	0.0506
Ax = b comm. time(s)	0.5856	0.0475	0.0015	0.0022
#messages	12	12	11	24
max non-zeros per proc.	4 346 286	4 319 031	4 431 469	4 264 282
max vector elems per proc.	345 727	381 623	501 669	557 602
max per proc. comm. vol.	1 326 626	147 078	2 110	1 901
total comm. vol.	2 646 280	223 467	2 428	3 018
$p = 8$	**Cyclic**	**GleZhu**	**1D hypergraph**	**2D hypergraph**
decomposition time(s)	Neg.	Neg.	280.9	11 360
#iterations	77	81	83	81
per iteration time(s)	0.8489	0.0756	0.0458	0.0444
Ax = b time(s)	0.7985	0.0641	0.0290	0.0292
Ax = b comp. time(s)	0.1455	0.0251	0.0276	0.0263
Ax = b comm. time(s)	0.6537	0.0395	0.0024	0.0028
#messages	56	56	46	105
max non-zeros per proc.	2 196 083	2 165 349	2 218 547	2 185 533
max vector elems per proc.	172 864	204 069	335 547	309 293
max per proc. comm. vol.	1 214 716	105 491	3 248	2 996
total comm. vol.	4 800 997	266 447	4 758	5 867
$p = 16$	**Cyclic**	**GleZhu**	**1D hypergraph**	**2D hypergraph**
decomposition time(s)	Neg.	Neg.	157.3	7 857
#iterations	74	81	79	80
per iteration time(s)	0.9548	0.0577	0.0396	0.0405
Ax = b time(s)	0.8755	0.0455	0.0229	0.0255
Ax = b comp. time(s)	0.2797	0.0207	0.0194	0.0198
Ax = b comm. time(s)	0.5987	0.0257	0.0045	0.0055
#messages	240	240	154	306
max non-zeros per proc.	1 124 363	1 126 092	1 110 174	1 091 597
max vector elems per proc.	86 432	122 143	182 236	198 703
max per proc. comm. vol.	928 783	88 210	4 486	3 896
total comm. vol.	7 237 257	313 198	14 433	11 684

Tab. 4. India Web graph Results

factors of 2 and 6 respectively. This reduction results in a decrease in the overall PageRank iteration time by 50% in the 2D case.

We note that, contrary to intuition, in some cases computation times do vary significantly depending on decomposition method used. We conjecture that this occurred because we did not make any attempt to optimise the caching behaviour of our parallel PageRank solver. As a consequence the GleZhu method (which assigned consecutive vector elements to processors) has a good cache hit rate; conversely the cyclic method (which assigned vector elements on a striped basis) suffers a poor cache hit rate.

5 Conclusion and Future Work

In this paper, we have sought to speed up the execution of parallel PageRank computation through the use of hypergraph partitioning-based decomposition techniques. We have investigated the application of both one- and two-dimensional hypergraph models, and compared them to conventional load balancing decomposition methods. Our experiments on a gigabit PC cluster have shown that hypergraph-based models consistently and substantially decrease distributed per-iteration communication overhead, resulting in the halving of per-iteration run-time when compared to the best available currently-used alternative.

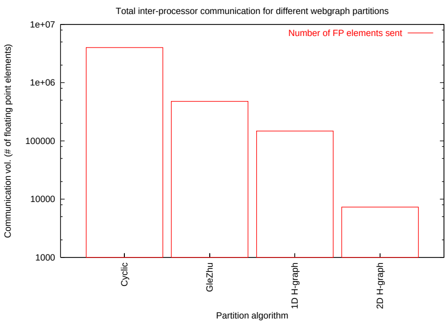

Fig. 3. Total per-iteration communication volume for 16-processor Stanford_Berkeley PageRank computation.

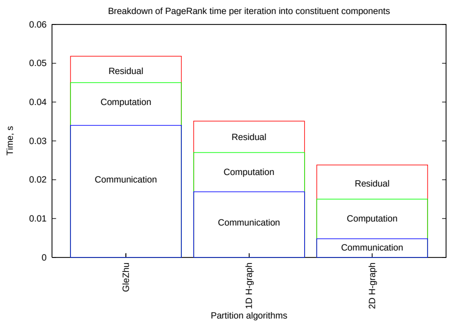

Fig. 4. Per-iteration execution time for 16-processor Stanford_Berkeley PageRank computation.

Because of the initial partitioning overhead, the proposed technique is particularly applicable when performing PageRank calculations with multiple personalisation vectors, since the same partition can be reused at no additional cost. We observed that the partitioning overhead was relatively low for the 1D hypergraph decomposition when compared to the 2D hypergraph decomposition. We have some observations to make about this. Firstly, the 2D hypergraph decomposition is a harder problem to solve, since the more sophisticated layout requires the solution of a much larger hypergraph partitioning problem instance with unique characteristics. Secondly, the parallel partitioning tool used (i.e. Par*k*way 2.1) is constantly evolving and has not yet been optimised for 2D decomposition. Furthermore, other emerging hypergraph partitioning tools (e.g. Zoltan [7]) promise potentially much faster parallel execution times, for both 1D and 2D decomposition.

In terms of future work, the current decomposition models aim to minimise total communication volume only. However, depending on the characteristics of the interconnection network used, performance may also be significantly affected by factors such as the number of messages sent or the maximum communication volume passing through a processor. To this end, we aim to develop hypergraph models which incorporate message and communication volume balancing constraints. Secondly, the 2D hypergraph-based decomposition gives rise to a hypergraph where each vertex is incident on exactly two hyperedges. Faster parallel partitioning algorithms may be developed, exploiting this favourable structure.

References

1. L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank citation ranking: Bringing order to the web,” Tech. Rep. 1999–66, Stanford Univ., November 1999.

2. T. H. Haveliwala, "Topic sensitive PageRank: A context-sensitive ranking algorithm for web search," Tech. Rep., Stanford University, March 2003.
3. C. Alpert, J.-H. Huang, and A. Kahng, "Recent Directions in Netlist Partitioning," *Integration, the VLSI Journal*, vol. 19, no. 1–2, pp. 1–81, 1995.
4. U. V. Catalyurek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse matrix–vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 7, pp. 673–693, 1999.
5. B. Vastenhouw and R. H. Bisseling, "A Two-Dimensional Data Distribution Method for Parallel Sparse Matrix-Vector Multiplication," *SIAM Review*, vol. 47, no. 1, pp. 67–95, 2005.
6. A. Trifunovic and W. J. Knottenbelt, "Par$k$way2.0: A Parallel Multilevel Hypergraph Partitioning Tool," in *Proc. 19th International Symposium on Computer and Information Sciences* (C. Aykanat, T. Dayar, and I. Korpeoglu, eds.), vol. 3280 of *Lecture Notes in Computer Science*, pp. 789–800, Springer, 2004.
7. E. Boman, K. Devine, R. Heaphy, U. Catalyurek, and R. Bisseling, "Parallel hypergraph partitioning for scientific computing," Tech. Rep. SAND05–2796C, Sandia National Laboratories, Albuquerque, NM, April 2005.
8. S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub, "Extrapolation methods for accelerating PageRank computations," in *Twelfth International World Wide Web Conference*, (Budapest, Hungary), pp. 261–270, ACM, May 2003.
9. D. de Jager, "PageRank: Three distributed algorithms," M.Sc. thesis, Department of Computing, Imperial College London, London SW7 2BZ, UK, September 2004.
10. A. N. Langville and C. D. Meyer, "Deeper inside PageRank," *Internet Mathematics*, vol. 1, no. 3, pp. 335–400, 2004.
11. T. H. Haveliwala and S. D. Kamvar, "The second eigenvalue of the google matrix," Tech. Rep., Computational Mathematics, Stanford University, March 2003.
12. "Google." `http://www.google.com/`. 20th June 2005.
13. S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub, "Exploiting the block structure of the web for computing PageRank," Stanford database group tech. rep., Computational Mathematics, Stanford University, March 2003.
14. D. Gleich, L. Zhukov, and P. Berkhin, "Fast parallel PageRank: A linear system approach," Tech. Rep., Institute for Computation and Mathematical Engineering, Stanford University, 2004.
15. U. V. Catalyurek and C. Aykanat, "A Fine-Grain Hypergraph Model for 2D Decomposition of Sparse Matrices," in *Proc. 8th International Workshop on Solving Irregularly Structured Problems in Parallel*, (San Francisco, USA), April 2001.
16. B. Ucar and C. Aykanat, "Encapsulating Multiple Communication-Cost Metrics in Partitioning Sparse Rectangular Matrices for Parallel Matrix-Vector Multiples," *SIAM Journal of Scientific Computing*, vol. 25, no. 6, pp. 1837–1859, 2004.
17. B. A. Hendrickson, "Graph partitioning and parallel solvers: Has the Emperor no clothes," in *Proc. Irregular'98*, vol. 1457 of *LNCS*, pp. 218–225, Springer, 1998.
18. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., 1979.
19. A. Trifunovic and W. Knottenbelt, "A Parallel Algorithm for Multilevel k-way Hypergraph Partitioning," in *Proc. 3rd International Symposium on Parallel and Distributed Computing*, (University College Cork, Ireland), pp. 114–121, July 2004.
20. T. Davis, "University of Florida Sparse Matrix Collection," March 2005. http://www.cise.ufl.edu/research/sparse/matrices.
21. "UbiCrawler project." `http://webgraph-data.dsi.unimi.it/`.