Scalable Computing for Power Law Graphs: Experience with Parallel PageRank

David Gleich*
Stanford University
ICME
Stanford, CA 94305
dgleich@stanford.edu

Leonid Zhukov
Yahoo!
701 First Ave
Sunnyvale, CA 94089
zhukov@yahoo-inc.com

ABSTRACT

In this paper we investigate the numerical and parallel performance of linear algebra algorithms when applied to power law data typical for information retrieval. In particular, we report on the behavior of parallel numerical algorithms applied to computing the PageRank vector for a 1.5 billion node directed web graph. We also describe the details of our parallel implementation that was able to compute the PageRank vector for this web graph on a distributed memory 140 processor RLX cluster in under 6 minutes.

Categories and Subject Descriptors

G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; D.2 [**Software**]: Software Engineering; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Performance, Experimentation

Keywords

PageRank, Eigenvalues, Linear Systems, Parallel Computing, Power Law, Distributed Memory, Sparse Matrix, Web Graph

1. INTRODUCTION

Historically, much of the work in parallel computing has been focused on the numerical solutions of problems arising from the engineering and physical sciences. The typical goal is to numerically solve a problem formulated in a continuous domain using an appropriate discretization. The discretization process usually results in two or three dimensional meshes. For convenience of further discussion we will consider these meshes as a special case of undirected graphs. The graphs used in these problems are either regular, e.g. a grid, or have slight variance in the number of nearest neighbors and distance to nearest neighbors compared with the average value. Most of the widely used parallel computing toolkits are devoted to solving such problems and follow this paradigm.

Numerical problems arising in data mining and information retrieval have quite different properties. First, the data is already discrete and there is no continuous representation possible. Thus, we have no ability to choose a discretization level and control the size of the computation. Second, there is no low-dimensional space easily associated with the data, thus algorithms that make use of the physical location of the elements are unusable, e.g. non-algebraic multi-grid algorithms. Nevertheless, the majority of the data can be represented as a graph. Datasets such as document collections and customer transactions can be represented as bipartite graphs. Also, the web graph is a directed graph, and so we will focus on graph properties to characterize the data. The third and most important difference from standard scientific computing datasets is the presence of a power law distribution in the graph. A power law graph is defined by the property that the number of vertices with degree k is proportional to $k^{-\beta}$ for the power law exponent β. In such a graph there are a few vertices with extremely high degrees and many vertices with low degrees. For example, in a web graph, the in-link degrees vary from 10,000, for sites like Yahoo and CNN, to 1, for most personal home pages.

The graphs arising in communication networks, social networks, biology networks, and text databases all have the power law property. One of the properties of a power law graph is that the average distance between nodes is small [9]. Another term widely used for a power law graph is a scale-free network [2]. Finally, some power law graphs also display the small-world phenomenon, for example, the social network formed by individuals. The small-world phenomenon can be interpreted as a power law distribution combined with a local clustering effect [10]. These two properties, small average distance and local clustering, are very different from the regularity of finite difference or finite element graphs.

In this paper, we investigate the use of existing parallel computing tools and their behavior on these new power law graphs. In particular, we will focus on web graphs and a parallel PageRank computation. This computation has the additional property that it uses a directed graph and non-symmetric matrix.

To perform multiple numerical experiments on real Web graph data, we developed a system that can compute results within minutes on Web graphs with one billion or more links. When constructing our system, we did not attempt to optimize each method and instead chose an approach that is

*Work performed while at Yahoo!

amenable to working with multiple methods in a consistent manner while still easily adaptable to new methods. An example of this trade-off is that our system stores edge weights for each graph, even though many Web graphs do not use these weights.

The remainder of the paper is organized as follows. Section 2 introduces the PageRank vector and two formulas to compute it. Section 3 discusses our distributed memory parallel implementation of PageRank and methods for distributing power law graphs. Section 4 describes the datasets used in the experiment and elaborates on the known structural properties of web graphs. Our results in Section 5 are divided into results on the convergence of the PageRank problem and the performance of the parallel computation.

2. PAGERANK

The PageRank algorithm, a method for computing the relative rank of web pages based on the Web link structure, was introduced in [25] and has been widely used since then. The PageRank model involves a directed graph of links (edges) between web pages (nodes), a teleportation coefficient c, and a teleportation vector v. One interpretation of a page's PageRank is the probability of finding a random surfer on that page when the surfer randomly follows links between pages and restarts at a page in v with probability $(1-c)v_i$. PageRank computations are a key component of modern Web search ranking systems. For a general review of PageRank computing see [21].

Until recently, the PageRank vector was primarily used to calculate a global importance score for each page on the web. These scores were recomputed for each new Web graph crawl. Recently, significant attention has been given to *topic-specific* and *personalized* PageRanks [15, 17]. In both cases one has to compute multiple PageRanks corresponding to various teleportation vectors for different topics or user preferences.

PageRank is also becoming a useful tool applied in many Web search technologies and beyond. For example, spam detection [14] and trust networks [19] all use PageRank-style calculations. In this setting many PageRanks corresponding to different modifications – such as graphs with a different level of granularity (HostRank) or different link weight assignments (internal, external, etc.) – have to be computed. For each technology, the critical computation is the PageRank-like vector of interest. Thus, methods to accelerate and parallelize these computations are important.

Traditionally, PageRank has been computed as the principle eigenvector of a Markov chain probability transition matrix using a simple power iteration algorithm. Various methods to accelerate the simple power iterations process have already been developed, including an extrapolation method, a block-structure method, and an adaptive method. See the general review [21] for additional details on these methods.

In this paper we consider the PageRank linear system formulation [3] and iterative methods for its solution. An efficient solution of a linear system strongly depends on the proper choice of iterative method and, for high performance solvers, the computation architecture as well. There is no best overall iterative method; one of the goals of this paper is to investigate the best method for the particular class of problems arising from PageRank computations on parallel architectures.

2.1 PageRank Algorithm

In this section, we present the mathematical formulation of PageRank both as an eigen-system and as a linear system. Consider a Web graph adjacency matrix A of size n by n with elements A_{ij} equal to 1 when there is a link from URL i to URL j and equal to 0 otherwise. Here $i, j = 1 : n$ and n is a number of distinct URLs or pages. For pages with a non-zero number of out-links $deg(i) > 0$, the rows of A can be normalized (made *row-stochastic*) by setting $P_{ij} = A_{ij}/deg(i)$. Assuming there are no dangling pages (vide infra) the PageRank x can be defined as a limiting solution of the iterative process

$$x_j^{(k+1)} = \sum_i P_{ij} x_i^{(k)} = \sum_{i \to j} x_i^{(k)}/deg(i). \tag{1}$$

At each iterative step, the process distributes page authority weights equally along the out-links. The PageRank is defined as a *stationary* point of the transformation from $x^{(k)}$ to $x^{(k+1)}$ defined by (1). This transfer of authority corresponds to the Markov chain transition associated with the random surfer model when a random surfer on a page arbitrarily follows one of its out-links uniformly.

The Web contains many pages without out-links, called dangling nodes. Dangling pages present a problem for the mathematical PageRank formulation. A review of various approaches dealing with dangling pages can be found in [11].

One way to overcome this difficulty, is to slightly change the transition matrix P to a truly row-stochastic matrix

$$P' = P + d \cdot v^T, \tag{2}$$

where d_i is the dangling page indicator, $d_i = 1$ if page i is dangling and 0 otherwise, and v is some probability distribution over pages. This model means that the random surfer jumps from a dangling page according to a distribution v. For this reason vector v is often called a *teleportation* distribution. The original PageRank authors used uniform teleportation $v_i = 1/n$, but acknowledged that other choices were possible. Later authors [15, 17] showed that certain choices of v led to topic-specific and personalized PageRank vectors.

The classic simple power iterations (1) starting at an arbitrary initial guess converge to a principal eigenvector of P under the two conditions specified in the *Perron-Frobenius* theorem. The first condition, matrix *irreducibility* or *strong connectivity* of a graph, is not satisfied for a Web graph, while the second condition, *aperiodicity*, is routinely fulfilled (after the first condition is satisfied). The easiest way to achieve strong connectivity is to add a small degree of teleportation to every page.

$$P'' = cP' + (1-c)ev^T, \quad e = (1, \ldots, 1)^T \tag{3}$$

where c is a teleportation coefficient. In practice $0.85 \leq c < 1$. After these modifications, the matrix P'' is row-stochastic and irreducible, and therefore, simple power iterations

$$x^{(k+1)} = P''^T x^{(k)} \tag{4}$$

for the eigensystem $P''^T x = x$ converge to the principal eigenvector with eigenvalue 1. Equation 4 is the eigensystem formulation of the PageRank problem.

Now, combining Eq.(2 - 4) we get

$$[cP^T + c(vd^T) + (1-c)(ve^T)]x = x. \tag{5}$$

Using $(e^T x) = (x^T e) = ||x||_1 = ||x||$ (henceforth, all norms will be assumed to be 1-norms) we can derive a convenient identity

$$(d^T x) = ||x|| - ||P^T x||. \quad (6)$$

Then, Eq. (5) can be written as a linear system

$$(I - cP^T)x = k \cdot v \quad (7)$$

where $k = k(x) = ||x|| - c||P^T x|| = (1-c)||x|| + (d^T x)$.

A particular value of k only results in a rescaling of x, which has no effect on page ranking. The solution can always be normalized to be a probability distribution by $x/||x||$. The non-normalized solution to the linear system has the advantage of depending linearly on the teleportation vector. This property is important for blending topical or otherwise personalized PageRanks.

Casting PageRank as a linear system was suggested by Arasu et al. [3] who considered the Jacobi, Gauss-Seidel, and Successive Over-Relaxation methods.

2.2 Numerical Solution

2.2.1 *Eigenvector: Power Iterations.*

As previously mentioned, the standard way of computing PageRank is to look for the dominant eigenvector of P''^T with eigenvalue 1. Using the identity in equation 6 we can derive a more efficient algorithm to compute $P''^T x$,

$$P''^T x = cP^T + (1 - c||P^T x||)v, \quad (8)$$

which gives rise to the standard PageRank algorithm [25].

$x^{(0)} = v$
repeat
  $y = cP^T x^{(k)}$
  $\gamma = 1 - ||y||_1$
  $x^{(k+1)} = y + \gamma v$
until $||x^{(k)} - x^{(k-1)}||_1 > \delta$.

The convergence of this algorithm is proportional to the second eigenvalue. For the matrix P'', Haveliwala showed that the second eigenvalue $\lambda_2 \leq c$, so power iterations converge with speed c [16]. We do not consider more advanced methods for computing the eigenvector such as ARPACK [22] because non-dominant eigenvectors are often complex valued for non-symmetric matrices, which significantly increases the computational difficulty.

2.2.2 *Linear System: Iterative Methods*

The PageRank linear system matrix $A = I - cP^T$ is large, sparse and non-symmetric. Solution of the linear system, Eq. (7), by a direct method is not feasible due to the matrix size and computational recourses. Sparse LU factorization [23] can still be considered for smaller problems (or subproblems), but it does create additional fill in. It is also notoriously hard to parallelize.

In this paper we concentrate on the use of iterative methods [13, 6]. There are two requirements for the iterative linear solver: i) it should work with nonsymmetric matrices and ii) it should be parallelizable. Because the matrix is strongly diagonally dominant, we use stationary Jacobi iterations. Additionally, we have chosen several Krylov subspace methods.

Jacobi iterations. The Jacobi process is the simplest stationary iterative method and is given by

$$x^{(k+1)} = cP^T \cdot x^{(k)} + b, \quad (9)$$

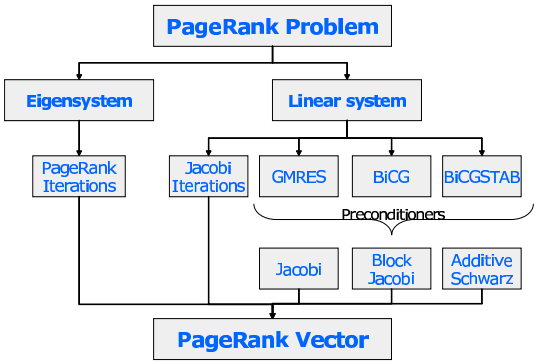


Figure 1: A flowchart of the computation methods used in this paper. The PageRank problem can be solved as both the dominant eigenvector or the solution of a linear system.

after appropriate substitutions for the PageRank problem.

Jacobi iterations formally result in the geometric series $x = \sum_{k \geq 0} (cP^T)^k b$, when started at $x_0 = b$ and thus, also have the rate of convergence of at least c. On graphs without dangling pages, the convergence is precisely c and Jacobi and power iterations result in the same algorithm.

Krylov subspace methods. We also consider a set of Krylov subspace methods to solve the linear system Eq. (7). These methods are based on certain minimization procedures and only use the matrix through matrix-vector multiplication. Detailed description of the algorithms are available in [6] and [4].

In this study we have chosen several Krylov methods satisfying our criteria:

- Generalize Minimum Residual (GMRES)
- Biconjugate Gradient (BiCG)
- Quasi-Minimal Residual (QMR)
- Conjugate Gradient Squared (CGS)
- Biconjugate Gradient Stabilized (BiCGSTAB)
- Chebyshev Iterations.

Although the convergence of these methods is less predictable than for Jacobi iterations, it can often be improved by the use of preconditioners. In this study we have used parallel Jacobi, block Jacobi, and additive Schwarz preconditioners.

In all of the above methods we start with the initial guess $x^{(0)} = v$. Note that the norm of $||x^{(k)}||$ is not preserved in the linear system iterations. If desired, we can normalize the solution to become a probability like the true PageRank vector by computing $x^{(k)}/||x^{(k)}||$.

Figure 1 presents a computation flowchart of all the methods used in this paper. The computational complexity and space requirements of these methods are given in Table 1.

Finally, to monitor the convergence of iterative methods, we used absolute error $||x^{(k+1)} - x^{(k)}||$ for power iterations and normalized residual error $||Ax^{(k)} - b||/||x^{(k)}||$ for the linear systems.

Method	IP	SAXPY	MV	Storage
PAGERANK		1	1	$M + 3v$
JACOBI		1	1	$M + 3v$
GMRES	$i + 1$	$i + 1$	1	$M + (i + 5)v$
BiCG	2	5	2	$M + 10v$
BiCGSTAB	4	6	2	$M + 10v$

Table 1: Computational Requirements. This table enumerates the operations per iteration. IP counts inner products, SAXPY counts AXPY operations, MV counts matrix vector multiplications, and Storage counts the number of matrices (M) and vectors (v) required for the method.

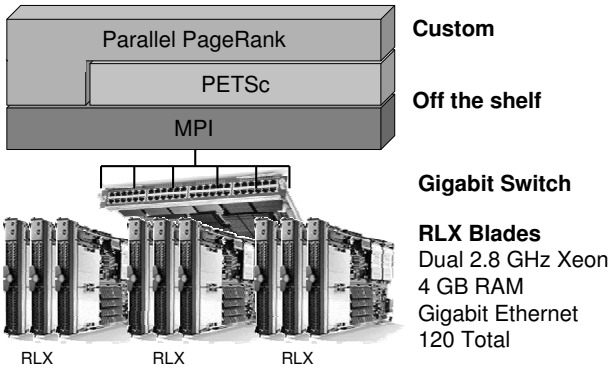


Figure 2: Our Parallel PageRank System.

3. PARALLEL IMPLEMENTATION

We chose a parallel implementation of the PageRank algorithms to meet the scalability requirement. We wanted to be able to compute PageRank vectors for large graphs (one billion links or more) quickly to facilitate experiments with the PageRank equation. To that end, our goal was to keep the entire Web graph in memory on a distributed memory parallel computer while computing the PageRank vector. An alternate approach explored in [24] is to store a piece of the web graph on separate hard disks for each processor and iterate through these files as necessary.

Our parallel computer was a Beowulf cluster of RLX blades connected in a fully connected topology with gigabit ethernet. We had twelve chassis composed of 10 dual processor Intel Xeon blades with 4 GB of memory each (240 processors, and 480 GB memory total). Figure 2 shows a visual representation of our parallel system.

The parallel PageRank codes use the Portable, Extensible Toolkit for Scientific Computation (PETSc) [4, 5] to implement basic linear algebra operations and basic iterative procedures on parallel sparse matrices. In particular, PETSc contains parallel implementations of many linear solvers, including GMRES, BiCGSTAB, and BiCG. While PETSc provided many of the operations necessary for working with sparse matrices, we still had to develop our own tools to load the Web graphs as parallel sparse matrices, distribute them among processors, and perform load balancing. We also implemented the optimized power iteration algorithm from Section 2.2.1 and Jacobi iterations.

PETSc stores a sparse matrix in parallel by dividing the rows of the matrix among the p processors. Thus, each processor only stores a submatrix of the original matrix. Further, PETSc stores vectors in parallel by only storing the rows of the vector corresponding to the matrix on that processor. That is, if we partition the n rows of a matrix A among p processors then processor i will store both the matrix rows and the corresponding vector elements. This distribution allows us to load graphs or matrices which are larger than any individual processor can handle, as well as to operate on vectors which are larger than any processor can handle. While this method allows us to work on large data, it may involve significant off-processor communication for matrix-vector operations. See [5] for a discussion of how PETSc implements these operations.

3.1 Load-Balancing Web Graphs

Ideally, the best way to divide a matrix A among p processors is to try to partition and permute A in order to balance work and minimize communication between the processors.

This classical graph partitioning problem is NP-hard, and approximate graph partitioning schemes such as ParMeTiS [20] and Pjostle [26] fail on such large power law data.[1]

Thus, we restricted ourselves to a simplified heuristic method to balance the work load between processors.

By default, PETSc balances the number of matrix rows on each processor by assigning each approximately n/p rows. For large Web graphs this scheme will result in a dramatic imbalance among the number of non-zero elements stored on each processor and often will result in at least one processor with more than 4 GB of data. To solve this problem we implemented a balancing scheme whereby we can choose how to balance the data among processors at run-time.

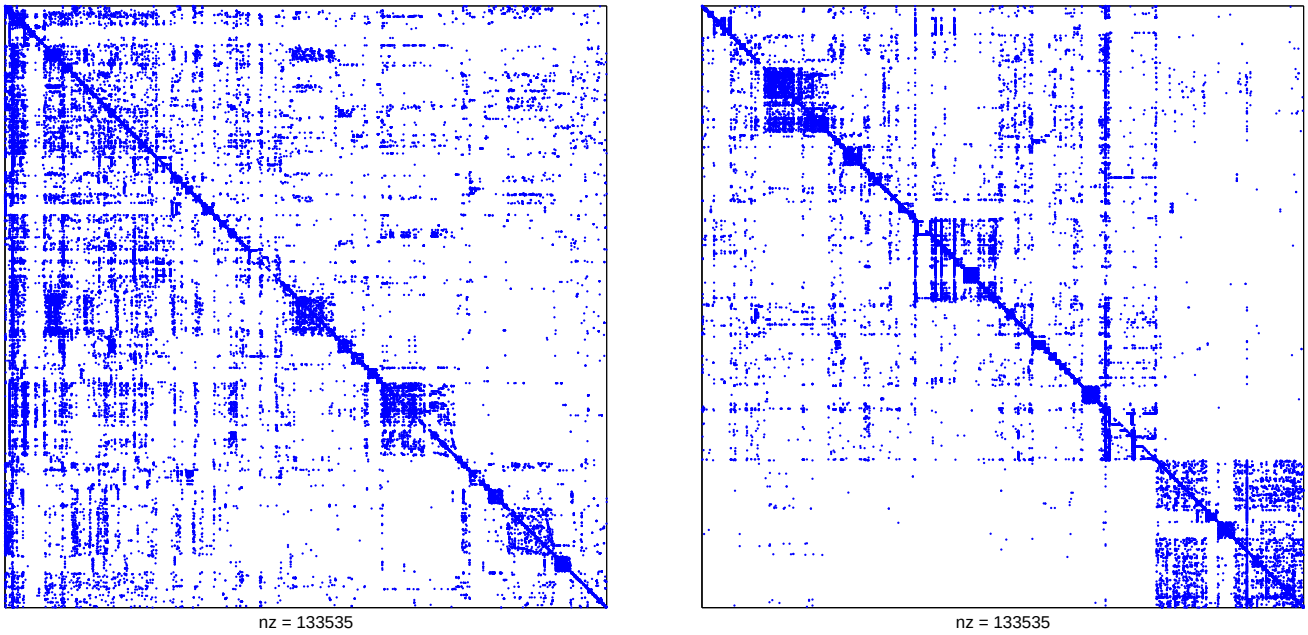
We allow the user to specify two integer weights w_{rows} and w_{nnz} and try to balance the quantity $w_{\text{rows}}n_p + w_{\text{nnz}}nnz_p$ among processors, where n_p and nnz_p are the number of rows and non-zeros on processor p, respectively. To implement this approximate balancing scheme, we always store entire rows on one processor and keep adding rows to a processor and incrementing a counter until this counter exceeds a determined threshold $(w_{\text{rows}}n + w_{\text{nnz}}nnz)/p$. Typically, we use equal weighing between rows and non-zeros ($w_{\text{rows}} = 1, w_{\text{nnz}} = 1$). Due to the large number of low-degree nodes in our power law dataset, this approximate scheme, in fact, gives us a good balance of work and memory among processors.

This loading scheme is far from ideal. We discuss one set of problems associated with loading power law graphs in general in Section 4.1. Second, we discuss a set of heuristics for working with our largest graph in the following section.

3.2 Method Specific Distribution

A few tweaks were required to allow the av graph (vide infra) to work with our implementation. This graph has 1.5 billion nodes and 6.6 billion edges. Even equal balancing between nodes and edges, $w_{\text{rows}} = 1$ and $w_{\text{nnz}} = 1$, is insufficient for such large graphs and overloads the 4 GB memory limit on some nodes. Thus, we attempted to estimate the optimal balance for each algorithm. From Table 1, the standard power iteration algorithm for PageRank requires storing the matrix and 3 vectors. Since the matrix is stored in compressed-row format with double sized floating point values, PETSc requires at least $(12 \cdot nnz + 4n) + 3 \cdot (8n)$ bytes.

[1]We believe these failures are due to problems with coarsening the graph in the presence of a power law distribution.



(a) Original ordering (b) Host order

Figure 3: The sparsity pattern for the adjacency matrix bs-cc.

Because there is some overhead in storing the parallel matrix, we typically estimate about $16nnz$ total storage for the matrix (i.e. 33% overhead). This yields a row-to-non-zero distribution of approximately 2 to 1. Thus, for simple power iterations we set $w_{\text{rows}} = 2$ and $w_{\text{nnz}} = 1$ for our dynamic matrix distribution. A similar analysis for the BiCGSTAB algorithm yields the values $w_{\text{rows}} = 4$ and $w_{\text{nnz}} = 1$.

4. DATA

In our experiments we used seven Web related directed graphs. The av graph is the Alta Vista 2003 crawl of the Web, the largest dataset used in our experiments. We also constructed and used four subsets of the av graph: the bs-cc graph is the largest strongly connected component of all hosts under `stanford.edu` and `berkeley.edu`; the edu graph contains web pages from the .edu domain; the yahoo-r2 and yahoo-r3 graphs contain pages reachable from `yahoo.com` by following two and three in- or out-links. We also used a Host graph, db, obtained from a Yahoo! crawl by linking hosts, e.g. `help.yahoo.com`, by agglomerating links between pages on separate hosts. We used an unweighted version of this graph for our experiments. Finally, the uk graph contains only UK sites and was obtained from [1]. Basic statistics for these graphs are given in Table 2.

Figure 3(a) shows a sparsity pattern of the matrix bs-cc. This matrix represents the connectivity pattern between pages on this small subset of the web. There are two important features of this matrix. First, it is unsymmetric. Second, there are patterns in the matrix: a few rows appear to have some block-structure (and thus, good parallel performance), but the left hand side of the matrix is fairly filled in. Without any good structure we often had to exchange data between all processors in an iteration.

The remainder of this section discusses a few aspects of the web graphs which are different from the regular graphs found in most scientific computing applications.

Name	Nodes	Links	Storage Size
bs-cc	20k	130k	1.6 MB
edu	2M	14M	176 MB
yahoo-r2	14M	266M	3.25 GB
uk	18.5M	300M	3.67 GB
yahoo-r3	60M	850M	10.4 GB
db	70M	1B	12.3 GB
av	1.4B	6.6B	80 GB

Table 2: Basic statistics for the data sets used in the experiments.

4.1 Node Degree Distribution

As discussed in the introduction, web graphs are power law graphs. The defining feature of a power law graph is that the number of nodes of degree k is proportional to $k^{-\beta}$. Previous studies of web graphs have shown that $\beta \approx 2.1$ [2]. Using the algorithm from [12], we calculated a power law coefficient near 1.5 for out-degrees and near 1.8 for in-degrees.

Figure 4 displays the power law distribution for the graphs bs, y2, and db. One interesting observation is that the db graph has a different out-degree distribution than the other graphs. Because this graph is formed by agglomerating pages on a single host together, this result makes sense. Although any particular page on the web may only link to a few others, essentially the out degree of any page is bounded; when pages on a host are agglomerated this bound is removed.

From an implementation perspective, the presence of the power law in the degrees of the graph poses a few challenges. First, we were not able to use standard tools such as ParMeTiS and pjostle to compute a distribution of the nodes over processors. These tools ran out of memory even

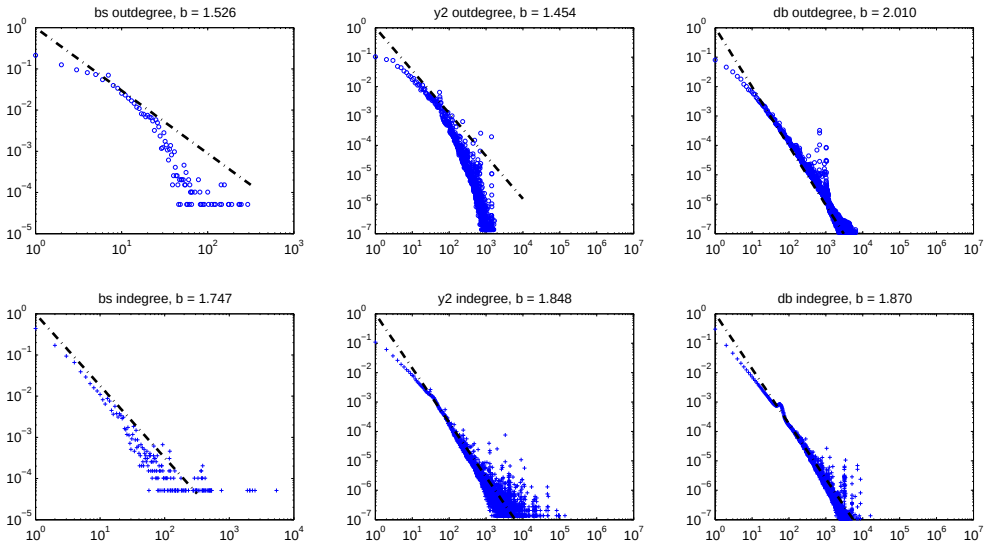


Figure 4: These plots show in-degree and out-degree distributions for three web graphs. Each plot is roughly linear on a log-log scale. The coefficient of the best fit line is listed as b in the title.

when it was plentiful (available memory was 10 times the size of the graph). Second, storing the references to off processor elements may be expensive. Because there is such diversity in the outlinks on the pages, a processor may have to store information on a large set of columns of the matrix.

4.2 Web Graph Structure

Broder el al. showed that the web graph has a "bow-tie structure" [7]. There is a large strongly connected component at the core of the web graph. Attached as the "bows" are large components of pages with only outlinks (left bow) and only inlinks (right bow). There are also a few long chains of pages.

Additionally, they note a few interesting results for random walks in the graph. Experimentally, they found that a random start and end node will only have a path between them 75% of the time. Further, the diameter of the graph is at least 500. These results were for a 1.5B link graph with 200M pages and we assume similar results are true for our graphs db and yahoo-r3.

Host Order. There is further structure in the web graph between hosts, e.g. `help.yahoo.com`, and pages [18]. Pages on a particular host will tend to link among each other and to common pages on that host. This effect can be considered the local clustering effect of the small world network in the web graph. Effectively, this result implies that the sparsity pattern of the web graph has a block structure associated with it. We can reveal the structure by permuting the matrix by a reverse domain ordering [18]. That is, we take a URL such as `help.yahoo.com/help/us/games` and reverse the order of the elements in the host name to get `com.yahoo.help/help/us/games`. We then sort the modified URLs lexicographically and permute the matrix accordingly. We refer to this ordering as the "host order" for the rest of the paper. Figure 3(b) shows the matrix bs-cc with the host ordering.

5. RESULTS

Our results are divided into two sections. In the first section, we present our results on the parallel performance of the algorithms. This section addresses issues such as scalability and load-balancing; i.e. how does the parallel implementation perform? The second section discusses the numerical performance of the algorithms. Here, we are primarily concerned with convergence in time; i.e. how long do these algorithms take to run?

5.1 Parallel Performance

5.1.1 *Distribution and Load-Balancing*

Our distribution and load balancing scheme (described in Section 3.1) allowed us to load and distribute the matrices to the cluster without overloading the memory on any particular machine. Figure 5 presents a plot of the runtime of the PageRank algorithm against time for one of the smaller graphs. For the simple distribution (an equal number of rows on each processor), the runtime displays oscillations as the number of processors increases. This behavior is smoothed by the load-balancing distribution. Because our algorithms are so sensitive to the communication pattern in the matrix-vector multiply operation, we believe that these peaks correspond to local maxima in the communication patterns. However, we never explicitly attempt to enforce the smoother behavior in our load-balancing and distribution; thus for other graphs, the load balancing distribution may also display the peaks.

5.1.2 *Scalability*

We have performed multiple numerical tests and present the results for the y3 and av graphs. We only present scalability results on a subset of the methods. (As we'll soon see

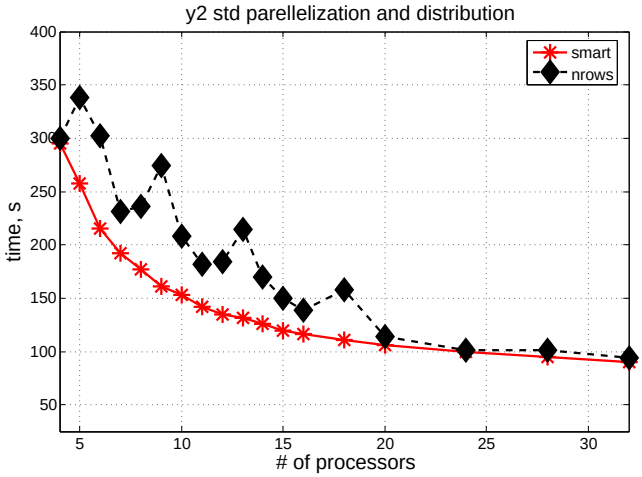


Figure 5: An illustration of the imbalance that occurs with different heuristic loading schemes. Smart balancing is 2 rows for every non-zero; nrows assigns an approximately equal number of rows to each processor. This figure shows oddities in the load-balancing scheme and how they appear in the run-time.

in the numerical experiments, these are the methods that matter – the best performing numerical algorithms.) Figure 6 presents the two scaling plots that show scalability of parallel PageRank. In both cases, the KSP solvers (GMRES and BiCGSTAB) scaled better than the power iteration algorithm. On the av graph, we showed "superlinear" speedup of the BiCGSTAB method. We believe that this large speedup is due to the simple distribution and load-balancing scheme employed. Effectively, the computations were done in the "peaks" in the distribution from Figure 5.

5.1.3 *Topology Experiments*

We also have performed some experiments with an alternate network topology. In the first configuration (called "star"), each of the 10 blades (20 processors) inside a chassis were connected to a single switch. The switches were then connected via a master switch. All the switches were gigabit speed. In the second configuration (called "full"), each blade was directly connected to a gigabit switch. The cluster topology had a dramatic effect on computation times. Because the latency is larger for the fully connected topology, we see a slight decrease in performance for our smallest graphs. For the larger graphs, the speedup is dramatic. On the av graph, the algorithms performed 3 times as fast on the fully connected topology.

The first number is the result from star topology; the second is from the full topology.

	Power	BiCGSTAB
av (140 procs)	2144 s/680 s	1982 s/955 s
db (60 procs)	806 s/557 s	315 s/220 s

5.2 Numerical Convergence Experiments

In this section, we'll focus on the performance of the algorithms from a numerical point of view. In this sense, the goal is to compute the PageRank vector to a given tolerance as quickly as possible. We have performed multiple experiments on convergence of the iterative methods on a set of graphs. We have studied the rate of convergence, the number of iterations and total time taken by a method to converge to a given accuracy. Our results are summarized in Table 3.

Graph	PR	Jacobi	GMRES	BiCG	BCGS
edu	84	84	$21^{\dagger}$	44^{*}	21^{*}
20 procs	0.09 s	0.07 s	0.6 s	0.4 s	0.4 s
yahoo-r2	71	65	12	35	10
20 procs	1.8 s	1.9 s	16 s	8.6 s	9.9 s
uk	73	71	22^{*}	25^{*}	11^{*}
60 procs	0.09 s	0.1 s	0.8 s	0.80 s	1.0 s
yahoo-r3	76	75			
60 procs	1.6 s	1.5 s			
db	62	58	29	45	15^{*}
60 procs	9.0 s	8.7 s	15 s	15 s	15 s
av	72				26
226 procs	6.5 s				16.5 s
av (host order)	72				26
140 procs	4.6 s				15.0

Table 3: Timing Results. See discussion in text.

In Table 3, the PR method is the standard PageRank power iteration method. The BCGS method is the BiCGSTAB method. The quasi-minimum residual (QMR), conjugate gradient squared (CGS), and Chebyshev methods did not converge on our data and we do not report additional results on these methods. In the table, the first line for each graph denotes the number of iterations required for each method to converge to an absolute residual value of 10^{-7}. The second line shows the mean time per iteration at the given number of processors. For the Krylov subspace methods, no superscript means we used no preconditioner, whereas $*$ denotes a block Jacobi preconditioner and $\dagger$ denotes an additive Schwarz preconditioner. We only report results for the fastest (by time) method. All Krylov solvers failed on "yahoo-r3" due to memory overloads on only 60 processors.

The convergence behavior of the algorithms on the uk and db graphs is shown in Figures 7 and 8. In these figures, we plot absolute error $||x^{(k+1)} - x^{(k)}||$ for power iterations and normalized residual $||Ax^{(k)} - b||/||x^{(k)}||$ for the linear systems.

We now discuss the results of our parallel linear system PageRank on a full Web graph. Other experiments with this data have been done by Broder et al. [8]. It is reported there that an efficient implementation of the serial PageRank algorithm took 12.5 hours on this graph using a quad-processor Alpha 667 MHz server. A similar serial implementation on a 800 MHz Itanium takes approximately 10 hours. Our main results on this graph are presented in Table 3 and the following table. These parallel run times do not include the time necessary to load the matrix into memory as repeated runs could be done with the in-memory matrix. By taking advantage of the block structure present in the data using the host ordering discussed in Section 4.2, we were able to decrease total computation time to 333 seconds.

	PR	BiCGSTAB
226 processors without host order	461 s	433 s
140 processors with host order	333 s	391 s

In these results, we see that power iterations is the fastest algorithm for 140 processors with the host ordering, but BiCGSTAB is the fastest method for 226 processors in the

original ordering. In many of our results, we saw these two methods interchange places. The relative performance of these algorithms is strongly dependent on the matrix-vector multiply time. If matrix-vector multiplications are "fast," then power iterations perform the best. Likewise, if matrix-vector multiplications are "slow," such as when we do not have a good method for distributing the matrix, then BiCGSTAB performs the best. Effectively, the extra work done by BiCGSTAB is only effective if it takes significantly less time than another matrix-vector multiplication. Because we observed that BiCGSTAB scales better than power iterations, as the number of processors increase, we anticipate this method to perform better than power iterations with many processors.

While the improvement due to the host order is impressive, it is not generically applicable. For example, the db graph is formed by agglomerating all the pages under one host. Hence, no host ordering exists for that graph.

In summary, our analysis of the results shows that:

- The power and Jacobi methods have approximately the same rate and the most stable convergence pattern. This is an advantage of stationary methods that perform the same amount of work per any iteration.
- The convergence of Krylov methods strongly depends on the graph and is non-monotonic.
- Although the Krylov methods have the highest average convergence rate and fastest convergence by number of iterations, on some graphs, the actual run time can be longer than the run time for simple power iterations.
- BiCGSTAB and GMRES have the highest rate of convergence and converge in the smallest number of iterations, with GMRES demonstrating more stable behavior.
- The best method to use is either power iterations or BiCGSTAB. The final choice of method is dependent on the time of a parallel matrix-vector multiply compared with the time of the extra work performed in the BiCGSTAB algorithm.

6. CONCLUSIONS

We have developed an efficient scalable parallel implementation for the solution of the PageRank problem and studied Jacobi and Krylov subspace iterative methods. Our numerical results show that power iterations and BiCGSTAB are overall the best choice of solution methods. Using our parallel implementation, we reduced the time to compute a PageRank vector on a full web graph from 10 hours to 5.5 minutes.

In developing our implementation, we discovered problems with tools typically associated with parallel computing. We were unable to compute a good partition for our dataset using ParMeTiS or pjostle. Further, due to the wide range of connections between elements in the directed web graph, a matrix vector multiply operation on our graph frequently involved all the processors communicating. Unfortunately, fully connected topologies such as the one used in this experiment are expensive and unscalable; thus, we need better ways of loading these graphs in a distributed fashion.

The number of power law graphs and the size of such graphs continues to grow. Current estimates of the full web graph size exceed 8 billion pages. As the size of such graphs grow, tools to efficiently work with them in parallel will become necessary. Continued work needs to be done to determine the best ways to manipulate these graphs on a distributed computer.

7. ACKNOWLEDGMENTS

We would like thank Farzin Maghoul for the Alta Vista web graph, Alexander Arsky for the Host graph, Matt Rasmussen, Pavel Berkhin, Jan Pedersen and Vivek Tawde for help and discussions and Yahoo! Research Labs for providing valuable resources and technical support.

8. REFERENCES

[1] Webgraph datasets. http://webgraph-data.dsi.unimi.it.

[2] R. Albert, A. Barabsi, and H. Jeong. Scale-free characteristics of random networks: The topology of the world wide web, 2000.

[3] A. Arasu, J. Novak, A. Tomkins, and J. Tomlin. PageRank computation and the structure of the web: Experiments and algorithms. In *WWW11*, 2002.

[4] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.

[5] S. Balay, V. Eijkhout, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[6] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[7] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. In *Proceedings of the WWW9 Conference*, pages 309–321, May 2000.

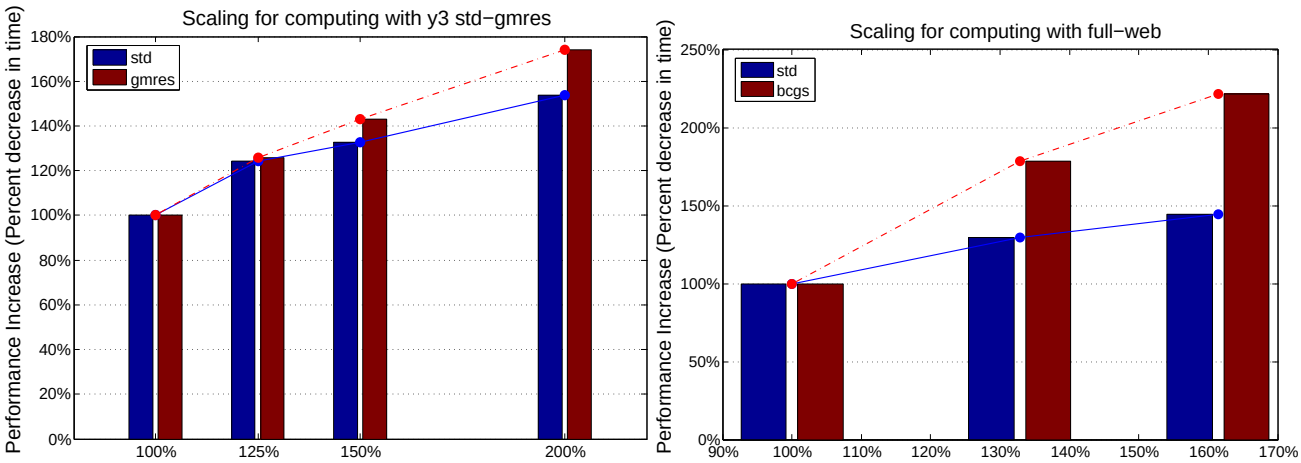
[8] A. Z. Broder, R. Lempel, F. Maghoul, and J. Pedersen. Efficient pagerank approximation via graph aggregation. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 484–485. ACM Press, 2004.

[9] F. Chung and L. Lu. Average distances in random graphs with given expected degree sequences. In *Proceedings of the National Academy of Science*, pages 15879–15882, 2002.

[10] F. Chung and L. Lu. The small world phenomenon in hybrid power law graphs. In *Lecture Notes in Physics*, pages 89–104, 2004.

[11] N. Eiron, K. McCurley, and J. Tomlin. Ranking the web frontier. In *WWW13*, 2004.

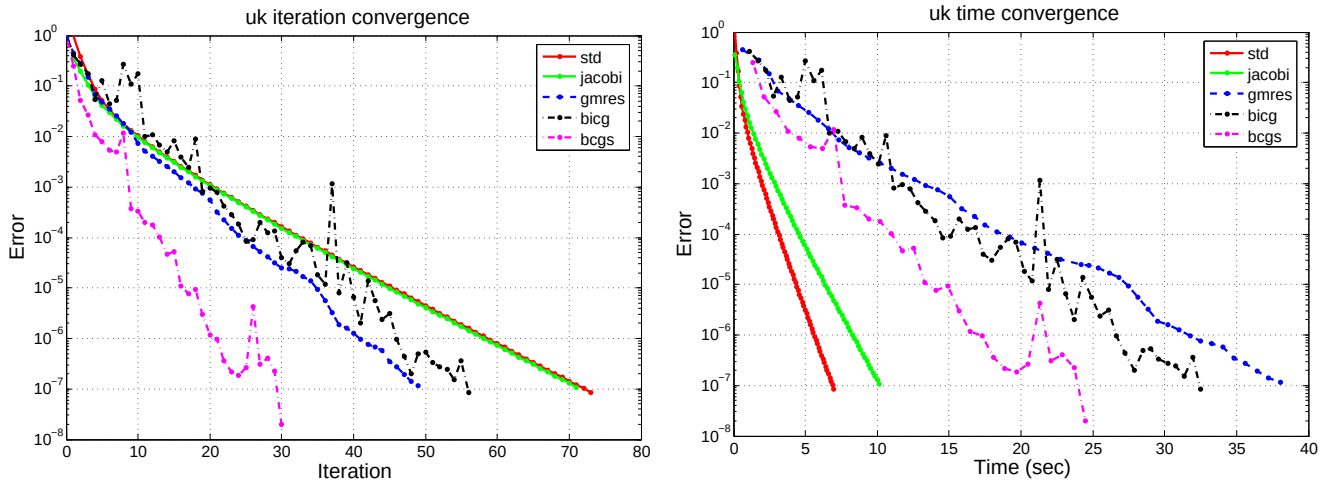
[12] M. L. Goldstein, S. A. Morris, and G. G. Yen. Problems with fitting to the power-law distribution. *European Physical Journal B*, 41:255–258, Sept. 2004.



(a) Power iterations and GMRES on y3.

(b) Power iterations and BiCGSTAB on av.

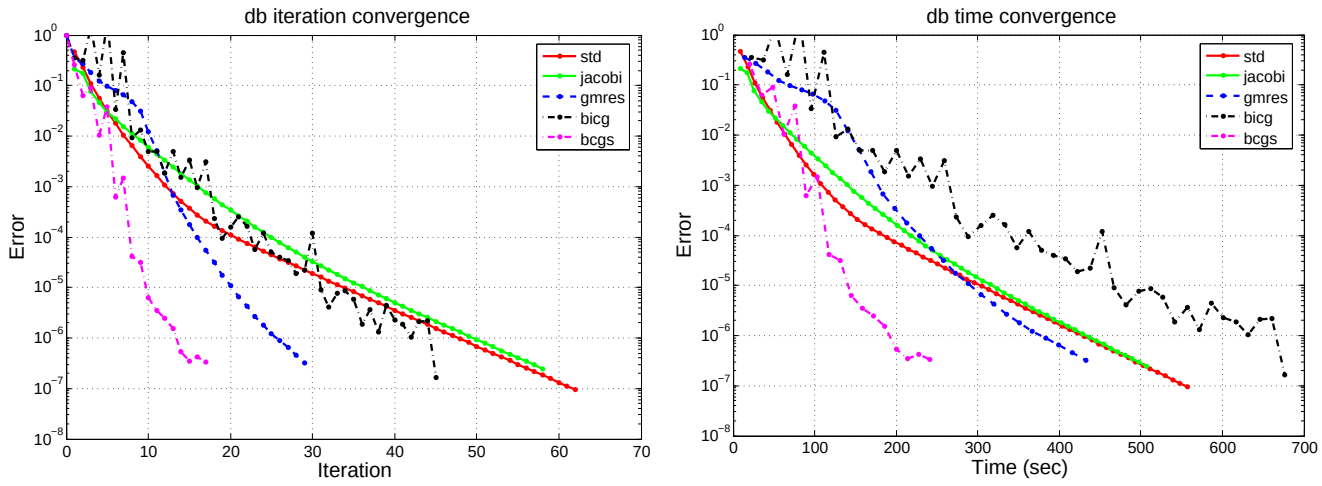
Figure 6: Parallel scaling performance of our algorithms. In both cases, we see that the KSP methods, BiCGSTAB (bcgs) and GMRES, scale better than power iterations (std).



(a) Convergence Iterations

(b) Convergence Time

Figure 7: Convergence of iterative methods on the uk Web graph.



(a) Convergence Iterations

(b) Convergence Time

Figure 8: Convergence of iterative methods on the db Host graph.

[13] G. H. Golub and C. F. V. Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, 1996.

[14] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *30th International Conference on Very Large Data Bases*, pages 576–587, 2004.

[15] T. Haveliwala. Topic-sensitive pagerank, 2002.

[16] T. Haveliwala and S. Kamvar. The second eigenvalue of the Google matrix. Technical report, Stanford University, California, 2003.

[17] G. Jeh and J. Widom. Scaling personalized web search. In *WWW12*, pages 271–279. ACM Press, 2003.

[18] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing pagerank, 2003.

[19] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[20] G. Karypis and V. Kumar. A coarse-grain parallel formulation of multilevel k-way graph-partitioning algorithm. In *Proc. 8th SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[21] A. N. Langville and C. D. Meyer. Deeper inside pagerank. Technical report, NCSU Center for Res. Sci Comp., 2003.

[22] R. Lehoucq, D. Sorensen, and C. Yang. *ARPACK Users' Guide: Solutions of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, 1997.

[23] X. Li and J. Demmel. Superlu dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems, 2003.

[24] B. Manaskasemsak and A. Rungsawang. Parallel pagerank computation on a gigabit pc cluster. In *Proceedings of the 18th International Conference on Advanced Information Networking and Applications*, 2004.

[25] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[26] C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 47(2):102–108, 1997.