A Scalable Parallel HITS Algorithm for Page Ranking

Matthew Bennett, Julie Stone, Chaoyang Zhang
School of Computing. University of Southern Mississippi. Hattiesburg, MS 39406
matthew.bennett@usm.edu, julie.stone@usm.edu, chaoyang.zhang@usm.edu

Abstract

The Hypertext Induced Topic Search (HITS) algorithm is a method of ranking authority of information sources in a hyperlinked environment. HITS uses only topological properties of the hyperlinked network to determine rankings. We present an efficient and scalable implementation of the HITS algorithm that uses MPI as an underlying means of communication. We then analyze the performance on a shared memory supercomputer, and use our results to verify the optimal number of processors needed to rank a large number of pages for the link structure of the total University of Southern Mississippi (usm.edu domain) web sites.

1. Introduction

Page ranking is an important part of information retrieval, related to searching through a group of web pages or other similar cross referenced resources. For example, when an internet search engine is used, a ranking is used to sort the resulting pages in terms of relevance and authority. Obviously, there is no absolute measure of the authority of a given resource, but approximations do exist [1]. It is important to have a constantly up-to-date compendium of page rankings, so that rankings relevant to a given search are no more than a few days old. Otherwise, the ranking is useless, since internet resources change so quickly, constantly replaced with other authoritative content.

This is problematic, since page ranking algorithms such as HITS [1] and Google's PageRank [8] are by their nature computationally expensive, and the number of accessible web pages which must be considered is incredible. One author estimated in January 2005 the total indexable number of web pages to be 11.5 billion [2]. Since internet growth has been explosively exponential, the total number of pages a major web search engine such as Google should index and rank will be tremendous. Page ranking in information retrieval is a good candidate for massively parallel systems that can constantly compute and cache large amounts of meta-data associated with information retrieval tasks.

The hypertext induced topic search (HITS) algorithm, developed by Kleinberg [1], utilizes the native link structures of a collection of web pages to determine the authority and connectivity of each. HITS assigns two attributes to a set of hypertext pages: Hubs and Authorities. A Hub value is a measure of the number of authoritative pages to which a page links. Each page has associated with it a real value $H\epsilon[0,1]$, which gives an indication of the quality and amount of links that page contains. An Authority value for a given page p is a measure of the quantity and Hub values of the pages that link to p. Each page has associated with it a real value $A\epsilon[0,1]$, which gives an indication of the quality and amount of links pointing to that page. Equations 1 can be used to calculate Authority values A and Hub values H, respectively.

$$A^p(v) = \sum_{(k,v)} H^{p-1}(k) \quad H^p(v) = \sum_{(v,k)} A^{p-1}(k) \qquad (1)$$

The A value at time step p for vertex v is given by the sum of the H values at time step $p-1$ of all pages which link to v. Similarly, the H value at time step p for vertex v is given by the sum of the A values at time step $p-1$ of all pages to which v links. Kleinberg suggests it is usually sufficient to terminate the algorithm and give rankings whenever $p \approx 20$. The algorithm is said to converge whenever the rankings at iteration k are identical to the rankings at iteration $k-1$ [1]. We use a graph to describe the link structure between pages. Internally, the link structure is represented as a standard unweighted adjacency matrix, in computer science texts such as [3].

2. Serial Algorithm

I Create Adjacency Matrix M. If there is a hyperlink from page A to page B, then the cell at M_{AB} is set to `true`.

II Calculate Hubs and Authorities. The initial values of hubs and authorities are the number of links from or to that node, respectively.

III Normalize Results so that the results gained map to $[0, 1]$.

IV Repeat II-III until the rankings stabilize.

V Sort Ranks. Sort out the ranks of the authorities to determine which page in the resultant set is the most relevant to this query.

Normalization of a set of either Hub or Authority values is done in the following way: Calculate the sum of the squares over all Hub or Authority values, then divide the square of each value by the sum of the squares. This may be done on a single machine in $\Theta(n)$ time, where n is the number of pages considered. For instance, the hub normalizations can each be calculated using equation 2 as suggested in [1].

$$h' = \frac{h}{\sqrt{\sum_{k \epsilon H} k^2}} \quad a' = \frac{a}{\sqrt{\sum_{k \epsilon a} k^2}} \tag{2}$$

3. Parallel Design

To calculate the authority value A_i^t for any given vertex i in our graph at time t, any implementation of the HITS algorithm needs to know all of the previous k hub values H_k^{t-1} for those vertices that link to vertex i. Similarly, to calculate the hub value H_i^t for vertex i at time t, HITS must know the previous authority values A_k^{t-1} for all k vertices which i links to.

If a processing element p is responsible for calculating all H_i and A_i for vertex i, then p needs to know all previous hub values for all vertices, and all previous authority values for all vertices, and also the network topology in the form of an adjacency matrix.

When the program begins execution, the adjacency matrix is read from disk at process 0. The input file can be in one of three file types: binary adjacency matrix, text adjacency matrix, or text adjacency list. The graph's internal representation is that of an adjacency matrix of Boolean values, which is flattened and broadcast to all processes. All processors receive the adjacency matrix along with its size. Each processing element is then delegated a number of vertices.

The number of vertices delegated to each process, known as $step$, is determined so that given n vertices and p processing elements, the first $p\ (n/p - \lfloor n/p \rfloor)$ of the processes are delegated $step = \lfloor n/p \rfloor$ vertices, and $p\ (1.0 - (n/p - \lfloor n/p \rfloor))$ are delegated $step + 1$ vertices. This process assures that the nodes are divided equally among processes. To assure that the division always works, the last process is always assigned the last vertex as a last element.

For the initial iteration of the HITS algorithm, no communication between nodes is necessary. The initial authority and hub sets can be calculated using only information available in the adjacency matrix. Each processing element calculates its designated hub values as row sums of the adjacency matrix, and calculates its designated authority values as column sums of the adjacency matrix. The processes then perform a Gather to process 0 in such a way that each process sends back only the Authority values for its designated set of vertices. An identical Gather occurs on the Hub values.

At this point, the master process may do one of two things. If the total number of iterations have not yet reached the desired number of iterations, the master process normalizes the sets of Hub and Authority values it has just gathered. It then broadcasts the Hub and Authority sets to all processes, so that they may begin the next iteration of the HITS algorithm. Otherwise, the master process sorts the vertexes into two new lists, by Authority weight and by Hub weight. These lists are the page rankings.

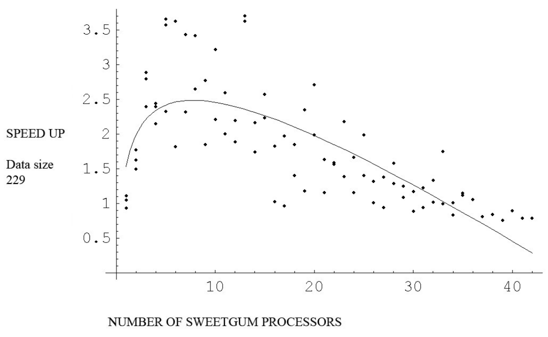

Figure 1. Speedup vs. processors for Block-and-broadcast on Sweetgum, input size 229

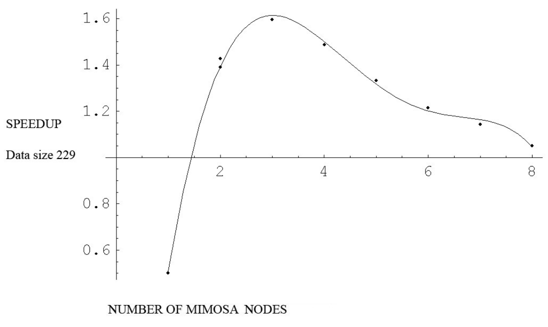

Figure 2. Speedup vs. processors for Block-and-broadcast on Mimosa, input size 229

Figures in 1 show the speedup obtained using the naive block-and-broadcast approach. (An explanation of speedup is given on page 4.) It is apparent that this approach is not scalable, as the speedup actually decreases when the number of processors is increased and the input size remains the same. There are two reasons for this: the idle time introduced by blocking, and also the communication cost associated with the non-buffered broadcasts at each iteration. All processors must wait to receive Hub and Authority broadcasts from a master single process before proceeding with the next iteration of HITS. The idle (blocked) time and communication cost of this approach imply a better communication pattern.

An alternative approach is to replace the {Gather to master process, normalize, Broadcast} method with the all-to-all Gather facility supported by MPI [4] and other message passing libraries. In this approach, each process is responsible for normalizing Hub and Authority values locally after each gather, and no special broadcast is needed from the first process. This saves significant communication and blocking time on massive problem sizes, and can be seen in the pseudo code. Communication is buffered, and occurs between all processes at the end of each iteration. Though the hub and authority vectors must be calculated locally, this can be done in substantially less time than the blocking time required for the previous method.

3.1. Parallel Algorithm

Driver

```
iterations := 20
curiter    := 0

//p0 load the file, then
//broadcast adj matrix
if rank := 0
   load file
   if size > 1
    Broadcast Adj Matrix

Split up the task evenly
 // (-> myFirst, myLast)

while next(curiter) < iterations
 HITS(adj matrix, n,
       myFirst, myLast, a, h)
 if curiter > 1
  normalize authority values
  normalize hub values
 All-Gather Authorities and Hubs

if (rank = 0)
   output the results
```

HITS (g, n, first, last, a, h)

```
sum :=  0
oldH[] :=  h
oldA[] :=  a

 //init the hubs, authorities
 static initialized = false
 if not initialized
  for y :=  0 to n-1
   sum :=  0
   for x :=  0 to n-1
     sum := sum + g[y*n + x]
   h[y] := sum

  for y :=  0 to n-1
   sum :=  0
   for x :=  0 to n-1
     sum :=  sum + g[x*n + y]
   a[y] :=  sum

 initialized :=  true

 else
  //Calculate Designated A
  for y in first to last
   sum :=  0

   for x in 0 to n - 1
     sum :=  sum + g[x*n+y]*oldH[x]
   a[y] :=  sum

  //calculate Designated H
  for y in first to last
    sum :=  0
    for x in 0 to n - 1
     sum :=  sum + g[y*n+x]*oldA[x]
    h[y] :=  sum

 return h,a
```

4. Development Environment

This project utilized the C++ programming language and the standard template library that is not present in the C language. The initial development of the serial algorithm occurred on stand-alone PCs, and then later ported over to MCSR Sweetgum [6] and MCSR Mimosa [7] for comparisons of the serial and parallel implementation. The testing of the parallel implementation occurred only on Sweetgum and Mimosa. All results were compared with each other to verify the results between the serial/parallel implementation and the implementation across platforms.

Sweetgum is an SGI Origin 2800, with 128 CPUs. 64 processors are 300MHz MIPS, and 64 are 195 MHz, with 64-bit words. As of this writing, Sweetgum runs IRIX 6.5.17, a variation of UNIX produced by SGI. The library used for message passing on Sweetgum was MPI-CH 1.2.1. [6]. MCSR Mimosa, a hybrid Linux cluster over Ethernet, was also used for comparison on small data sets [7].

5. Analysis

Calculate Hubs and Authorities	$2n^2$
Normalize Hubs, Authorities	$2n$
Repeat i times	$i * (prev\ steps)$

Table 1. Analysis for serial HITS algorithm

1 Broadcast the adjacency matrix	n^2t_c
2 Calculate Hubs and Authorities	$2\frac{n^2}{p}$
3 Normalize Hubs, Authorities	$2n$
4 Gather Hubs, Authorities	$2nt_c$
5 Repeat from 2 i times	$i * (prev\ steps)$

Table 2. Analysis for general parallel HITS algorithm

$$T_s = 2i(n^2 + n) = \Theta(in^2) \tag{3}$$

$$T_p = n^2t_c + 2i(\frac{n^2}{p} + nt_c + n) = \Theta((i+1)\frac{n^2}{p}) \approx \Theta(i\frac{n^2}{p}) \tag{4}$$

Loading the file and sorting the final results are inherently serial processes, which occur only at process 0. They are not included in our analysis, because they are common to both the serial and parallel implementations. If the first term (initial broadcast / start-up cost) is ignored, speedup S and efficiency E are approximated by

$$S = \frac{T_s}{T_p} = \frac{(1+n)p}{n+p+pt_c} \tag{5}$$

.

$$E = \frac{S}{p} = \frac{(1+n)}{n+p+pt_c} \approx \frac{n}{n+p(1+t_c)} = \frac{1}{1+(1+t_c)\frac{p}{n}} \tag{6}$$

.

This is scalable for large n, since $Limit\ n \rightarrow \infty\ S = p$. The definitions of scalability, efficiency, and speedup are provided by Grama, et al [5]. In equation 5, we can manipulate p for any given n so that $\frac{p}{n}$ remains constant. This analysis does not account for architectural limitations. Note also that if the number of iterations is small, the start-up cost is much more significant in the analysis. Kleinberg [1] places the recommended number of iterations before convergence between 20 and 100 for large sets of pages, so we do not consider the start-up term.

1 Broadcast the Adj Matrix	$(t_s + t_wbn^2)log(p)$
2 Calculate H, A values	$2\frac{n^2}{p}$
3 Normalize H, A values	$2n$
4 Gather H, A	$2n\log(p)(t_s + t_wn)$
5 Repeat from 2, i times	$i * (previous\ steps)$

Table 3. Analysis for parallel algorithm on a generalized architecture

$$T_p = 2i(\frac{n^2}{p} + 2t_slog(p) + t_wdn(p-1+log(p)+n) \tag{7}$$

The speedup associated with parallelization of the HITS algorithm comes from reducing the number of operations involved in the calculation of hubs and authority values. Assuming that each process is delegated the same number of vertexes, each process calculates its share of $2n/p$ values in parallel. In order for the algorithm to be efficient, we find the relationship between the number of processes p and the number of vertices n which minimizes T_p. Let t_w be the time to transfer one item of data, t_s be the start-up time, p be the total number of processors and n be the total number of vertices. Solving this equation 8 for p gives the optimal number of processes for graph with n vertices.

$$\frac{dT_p}{dp} = 2i(\frac{n^2}{p^2} + 2t_s\frac{1}{p} + nt_w(1+\frac{1}{p}) = 0 \tag{8}$$

Ignoring the leading constant, and solving for $p(n)$, we obtain 9. Choosing p processors as a function of the number of pages n to rank will optimize the total parallel run time (wall time) of the ranking. Note that no matter what the input size is, given an infinite parallel system, we can choose some number of processors that gives an optimal run time.

$$\rightarrow \frac{n^2}{p^2} + nt_w + \frac{1}{p}\,(2t_s + nt_w) = 0 \tag{9}$$

$$\rightarrow p(n) = \lceil \frac{-2t_s - nt_w + \sqrt{-4n^3t_w + (2t_s + nt_w)^3}}{2nw} \rceil \tag{10}$$

6. Results

We used a web spider to create an adjacency list for each of approximately 13500 interlinked web pages in the University of Southern Mississippi (usm.edu) domain. An adjacency list (or matrix) is the only input for our algorithm, as

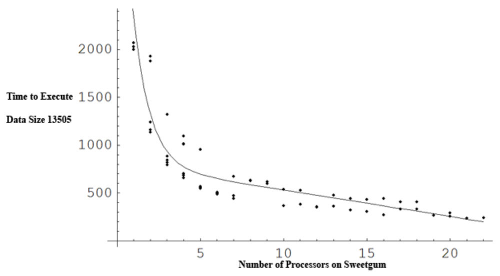

Figure 3. Wall time vs processors on sweetgum w/ real-world input size of n=13505 over five runs.

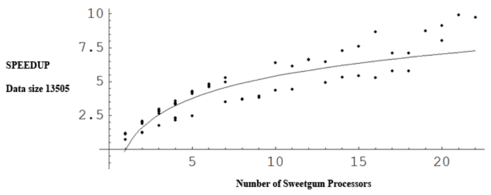

Figure 4. Speedup vs processors on sweetgum w/ real-world input size of n=13505 over five runs.

requires solely topological features as input. The total run time for our data set of 13500 nodes was a negative exponential function of the number of processors. Speedup was expectedly logarithmic.

For figure 3, the curve fit is $807.65 + -27.68x + 4225.78e^{-x}$, and is averaged over five data sets for 1 to 24 processes. The average serial time over this data set was 2350.32 seconds. The curve fit shown in figure 4 is $2.34729Lg(x)$, where Lg is the natural logarithm, as expected by our analysis. Note that in Figure 4, the graph is monotonically increasing, which is a property of a scalable algorithm.

7. Conclusion

In this paper, we have presented a parallel implementation of the HITS page-ranking algorithm [1]. We show graphically that a naive block-and-broadcast, centralized implementation is not scalable. We then improved the algorithm using the all-to-all gather communications pattern.

Analysis showed that the algorithm is scalable (for many iterations), and the results obtained for a large network of constant size support our analysis. Future work will provide a three-variable numerical result graph of speedup vs number of vertices and number of processes, showing isoefficiency lines. The algorithm can also be adapted to massively parallel networks, perhaps three orders of magnitude larger than the medium sized page network used to obtain results. Collecting this data itself in an ethical manner is time consuming, as many web sites prefer to spiders to not index them. Some performance tweaks may also be required to make the algorithm run in near-asymptotic time. Since Google Page Rank [8] works in a similar manner to the HITS algorithm, parallelization may be appropriate. This algorithm is not context-free like HITS[1], which may yield special challenges.

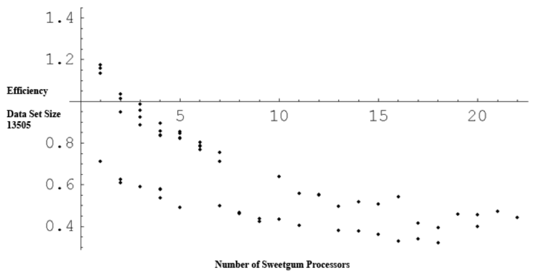

Figure 5. Efficiency vs processors on sweetgum w/ real-world input size of n=13505 over five runs.

8. Acknowledgement

We would like to thank the Mississippi Center for Supercomputing Research at the University of Mississippi for providing the computing resources used in obtaining our results [6], [7], and for providing support services.

References

[1] Kleinberg, J. **Authoritative Sources in a Hyperlinked Environment.** Journal of the ACM, Vol. 46, No. 5, September 1999, pp. 604–632.

[2] Gulli, A. Signorini, A. **The indexable web is more than 11.5 billion pages.** International World Wide Web Conference. New York: ACM. January 2005. ISBN 1595930515.

[3] Cormen T. Leiserson C. Rivest R. Stein, S. **Introduction to Algorithms**, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0262032937. Section 22.1: Representations of graphs, pp. 527531.

[4] Gropp, W. Lusk, E. Skjellum, A. **Using MPI: Portable Parallel Programming with the Message Passing Interface**. 2001. Cambridge: MIT. Scientific Engineering and Computation Series. ISBN 0262571323. Chapter 6: Advanced Message Passing in MPI, pp 121-131.

[5] Grama, A. Gupta, A. Karypis, G. Kumar, V. **Introduction to Parallel Computing**. 2nd Edition. January 16, 2003. Essex: Addison Wesley. ISBN 0201648652.

[6] MCSR. **SGI Origin 2800 at sweetgum.mcsr.olemiss.edu** Acessed 12-05-05 from http://www.mcsr.olemiss.edu/computing/sweetgum.html

[7] MCSR. **Beowulf Cluster at mimosa.mcsr.olemiss.edu** Acessed 12-05-05 from http://www.mcsr.olemiss.edu/computing/mimosa2.html

[8] Google Corporation. **Google searches more sites more quickly, delivering the most relevant results.** Acessed 05-18-06 from http://www.google.com/technology