Parallelization of Arnoldi algorithm for page rank computing by Cuda

Siddesh Sawant[1], **Pranjal Patidar**[1], **and Harish Kumar Datla**[1]

[1]M.tech, CSE, IIIT Hyderabad

ABSTRACT

In this paper we deal with parallel implementation of Google's PageRank algorithm. We used an iterative algorithm called Arnoldi method. The algorithm has been implemented in a parallel environment and has been tested on a weighted graph which was randomly generated. The parallel language we used is Cuda.

Introduction

Search engines are the centerhouses of web, serving people with information search results. Google,the most successful search engine in recent years is successful because of quick,comprehensive and accurate search results. For the search results to be quick we can see the use case for parallelizing the algorithm. The most dominant algorithm for ranking the search results is called PageRank. Google uses Power method for computing PageRank. This is the only method for the internet and larger domains due to its less memory requirements. But however for smaller domains Arnoldi method is a good candidate algorithm to investigate which converges after less iterations but has very high memory requirements. To handle these large-scale computations efficiently we need to implement the algorithms in parallel system.

PAGERANK REVIEW

The world wibe web can be imagined as a large graph with web pages as nodes and the links as edges of the graph. To simplify we assume the links(edges) are directed. To rank the importance of each specific page with respect to a search key, Google uses PageRank algorithm. This algorithm decides whether a specific page is important and how high it will show up in the search results.

a page is important if other important pages link to it, This is the gist of the PageRank algorithm. The links by lower rank pages doesn't contribute much for the rank. Also each link is weighted by the total no of links by a single page.

For n pages $P_i, i = 1,2,\ldots,n$ the corresponding PageRank is set to $r_i, i = 1,2\ldots,n$. PageRank can be recursively defined by following mathematical formulation.

$r_i = \sum_{j \in L_i} r_j/N_j, \quad i = 1,2,\ldots,n$

Where r_i is the PageRank of page P_i, N_j is the number of outlinks from page P_j and L_i are the pages that link to page P_i.

The above recursive formula can be implemented by an iterative code, but it needs several iterations before stabilizing to an acceptable solution. The iterative algorithm is as follows.

PageRank Algorithm :

1 : $r_i^{(0)}, \quad i = 1,2\ldots,n.$
2 : for k=0,1, ... do
3 : $\quad r_i^{(k+1)} = \sum_{j \in L_i} \frac{r_j^{(k)}}{N_j}, \quad i = 1,2\ldots,n$
4 : $\quad$ if $\left\| \mathbf{r}^{(k)} - \mathbf{r}^{(k+1)} \right\|_1 < tolerance$ then
5 : $\quad\quad$ break
6 : $\quad$ endif
7 : endfor

We initially start with an arbitrary guessed vector r that describes the initial PageRank value r_i for all pages P_i. Then you iterate over the recursive formula until two consecutively iterated PageRank vectors are similar enough.

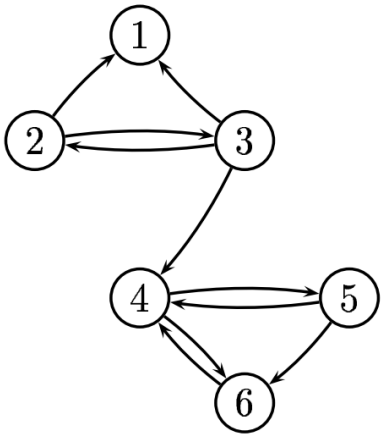
You start with an arbitrarily guessed vector r (e.g. a vector of ones, all divided with number of pages present), that describes the initial PageRank value r_i for all pages P_i. Then you iterate the recursive formula until two consecutively iterated PageRank vectors are similar enough.

Matrix Model

By defining a matrix

$$Q_{ij} := \begin{cases} 1/N_i & \text{if } P_i \text{ links to } P_j \\ 0 & \text{otherwise} \end{cases}$$

We can formulate the PageRank algorithm as a matrix-problem. For sample look at the below sample graph with only 6 web pages,$P_1, P_2, \ldots, P_6$.



If the above graphs link structure is converted to matrix formulation, We get the following matrix.

$$\mathbf{Q} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Here Q_{ij} means there is a link from P_i to page P_j, and these are all divided by the number of outlinks on page P_i represented as N_i.

The iteratively calculated PageRank r could then be written as:

$\mathbf{r}^T_{(k+1)} = \mathbf{r}^T_{(k)}\mathbf{Q} \quad , k = 0, 1, \ldots$

This is called power method.

We now take this matrix Q and now change in accordance to random walker model of the web by taking care of cases like stuck at a page and stuck in a subgraph cases we get

$\hat{\hat{\mathbf{Q}}}^T\mathbf{r} = \mathbf{r}$

Eigenvector computing

From the above equation we can see that PageRank is same as the eigenvector corresponding to the largest eigenvalue of the matrix $\hat{\hat{\mathbf{Q}}}^T$. Since mostly the World wide web is sparse, we need an iterative method that works well for large sparse matrices.

The Power method is old and obsolete method. However on the flip side it requires low memory, because it needs only one vector except for the unmodified matrix. Hence it is used in google.

0.1 Power Method

The Power method is a simple method for finding the largest eigenvalue and corresponding eigenvector of a matrix. It can be used when there is a dominant eigenvalue of A. That is, the eigenvalues can be ordered such that $\lambda_1 > \lambda_2 \geq \lambda_3 \geq \ldots \geq \lambda_n \cdot \lambda_1$ must be strictly larger than λ_2 that is in turn larger than or equal to the rest of the eigenvalues

Power method algorithm

1 : $\mathbf{x}_0$ = arbitrary non-zero starting vector
2 : for $k = 1, 2, \ldots$
3 : $\quad \mathbf{y}_k = \mathbf{A}\mathbf{x}_{k-1}$
4 : $\quad \mathbf{x}_k = \mathbf{y}_k / \|\mathbf{y}_k\|_1$

5 : end for

Since the matrix used by PageRank is very large, very few methods can successfully be used to calculate PageRank. The Power method is one of them and has some very good qualities which makes it a good option for google.

1) We only need to save the previous approximated eigenvector.

2) It finds the eigenvalue and vector for the largest eigenvalue, which is what we are interested in.

3) It does not in any way alter our matrix.

There are however better method that can be useful and effective when we are only interestes in a small subset of the enitre internet, say a university's intranet. With these smaller matrices, we can use more memory for our calculations. Arnoldi method is one such algorithm which is very well suited for these instances. We will discuss arnoldi next.

0.2 Arnoldi algorithm

Arnoldi method can be used to iteratively to find all the eigenvalues and their corresponding eigenvectors for a matrix A. It was first created and used for transforming a matrix into upper Hessenberg form but it was later seen that this method could successfully be used to find eigenvalues and eigenvectors for a large sparse matrix in an iterative fashion. **Arnoldi Method**

1 : $\mathbf{v}_0$ = arbitrary non-zero starting vector
2 : $\mathbf{v}_1 = \mathbf{v}_0 / \|\mathbf{v}_0\|_2$
3 : for $j = 1, 2, \ldots$ do
4 : $\quad \mathbf{w} := \mathbf{A}\mathbf{v}_j$
5 : $\quad$ for $i = 1 : j$do
6 : $\quad\quad h_{ij} = \mathbf{w}^* \mathbf{v}_i$
7 : $\quad\quad \mathbf{w} := \mathbf{w} - h_{ij}\mathbf{v}_i$
8 : $\quad$ endfor
9 : $\quad h_{j+1,j} = \|\mathbf{w}\|_2$
10: $\quad$ if $h_{j+1,j} = 0$
11: $\quad\quad$ stop
12: $\quad$ endif
13: $\quad \mathbf{v}_{j+1} = \mathbf{w}/h_{j+1,j}$
14: endfor

The decision to stop iterations at each step is based on the residual norm of the PageRank vector. When the residual between two consecutive iterations changes less than a certain tolerance we stop iterating. In the Arnoldi method instead of directly calculating the residual of two consecutive eigenvectors, we can use a very computationally cheap method for a stopping criterion. This method is very inexpensive and hence one of the advantages of Arnoldi method.

Now to apply arnoldi to PageRank the following iterative algorithm works. The iterative algorithm is for finding a specific eigenvalue and eigenvector which is effectively PageRank.

Initial :

- Create an initial basis, usually uniform.

For $m = 1, 2, \ldots$

- Add an extra basis to your subspace.
- Calculate the eigenvector/eigenvalue we are interested in from the Hessenberg-matrix.
- If $h_{m+1,m} \left| \mathbf{e}_m^* \mathbf{y}_i^{(m)} \right| <$ tol Break

Final :

- Find the corresponding eigenvector in the real matrix as in equation.

As the number of iterations increases, the amount of work required for the Arnoldi method increases rapidly. To deal with this we use the idea of explicit restart. In explicit restart, we perform m number of iterations and compute the PageRank vector, we end of we are satisfied of the results else we restart with initial vector as the current PageRank vector.

Parallel Implementation

To make PageRank-calculation efficient, we need to execute the algorithm in a parallel system. This opens up several possibilities in partitioning the data and load balancing the data. . When we try to load balance and partition the data there are several issues that must be weighted together, for example a good partitioning for one specific operation might give us problems for others.

The most expensive operations done in the calculation of the PageRank-values are matrixvector multiplications, and it is a perfectly parallel operation with several possible methods of partitioning both the matrix and the vector. We have considered three different methods for partitioning the link matrix among the processors.

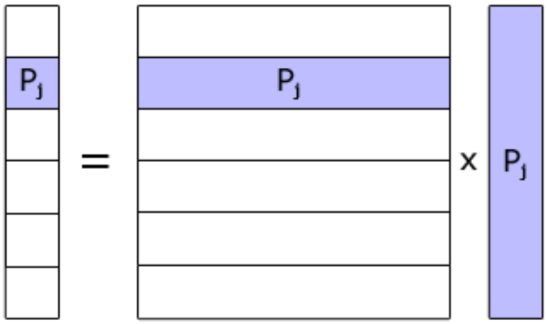
- Divide the matrix using a row-wise distribution.
- Divide the matrix using a column-wise distribution.
- Divide the matrix as a 2D cartesian grid.

The method chosen for our computations is the row-wise partitioning and this is because the matrix itself is stored in a (sparse) row-wise format, and any efficient partitioning must utilize the underlying storage-structure.

This also means that all processors have their own small part of the vector that is calculated in this matrix-vector multiplication. All these parts must then be gathered together by all processor to build the complete vector that was calculated in this multiplication. The vector used in the multiplication can also be divided among the processors, in several ways:

- Don't divide the vector at all, each processor holds a full copy of the vector we are multiplying with.
- Divide the vector into parts to go with the row-wise partitioning.

The method used when we calculate PageRank is the first one, as our problems are quite small in comparison and hence memory is not an issue and this saves time in communication.



Thus our iterative method for doing consecutive matrixxvector multiplications is
Start :

- Distribute the rows of the Matrix in some fashion
- All processors calculate the initial vector to multiply with.

Loop :

- All processors calculate their part of the result by multiplying their part of the matrix with the full vector that they have.
- All processor gather the new resulting vector and use it as the vector to multiply with in the next iteration.

Results

For numerical experiments we have implemented our algorithms in C++. For parallel environment we have used Cuda.

In our project we parallelized at steps 2,4,6,7,9,13 of the Arnoldi algorithm. We created a random 4X4 matrix and ran both serial and parallel code for different values of k = 4,8,16,32.

Here are the results

The time show in the table is in micro seconds.

k	serial	parallel	speedup
4	132k	110k	1.2
8	274k	163k	1.7
16	607k	333k	1.83
32	1238k	782k	1.6

As you can see there is almost 2x speedup in the parallel code than the serial code.

Conclusions

We have introduced and explored the performance of an algorithm based on the Arnoldi method for computing PageRank. It is a variant of the refined Arnoldi method which computes PageRank without using Ritz values. The strength of Arnoldi-type methods lies in obtaining orthogonal vectors. It allows for effective separation of the PageRank vector from other approximate eigenvectors. The algorithm shows robust performance.

References

1) An Arnoldi-type algorithm for computing page rank,G.H.Golub and C.Greif,Springer 2006
2) Investigating Google's PageRank algorithm,Erik Andersson and Per-Anders Ekström, 2004.