CS301 - High Performance Computing

Autumn 2017-18

PROJECT REPORT

Dwarf 2 - Sparse Linear Algebra and its Application in Page Rank Algorithm

Course Instructor : Prof. Bhaskar Chaudhury

Jay Goswami - 201501037
Vishalkumar Shingala - 201501450

Contents

1 Introduction

1.1 Sparse Linear Algebra

Sparse Linear Algebra is Dwarf 2 of among seven dwarfs for parallel computing. (dwarfs are key algorithmic kernels in many scientific computing applications) A matrix is called sparse, if it contains that many zero elements (i.e., that few non-zeros), such that it becomes worthwhile to change to special data structures and algorithms to store and process them.

Since special structures are used to store them, the algorithms like multiplying the matrix with a vector, transpose of a matrix requires different technique as compared to the traditional matrix multiplication and transpose algorithms.

Web matrix as Sparse Matrix

The webgraph describes the directed links between pages of the World Wide Web. The webgraph is a directed graph, whose vertices correspond to the pages of the WWW, and a directed edge connects page X to page Y if there exists a hyperlink on page X, referring to page Y. The webgraphs are generally sparse, since for each page, the count of pages having hyperlinks to the page, is very small as compared to the number of total webpages.

1.2 Page Rank Algorithm

PageRank (PR) is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages.

PageRank is a link analysis algorithm and it assigns a numerical weighting to each element of a hyperlinked set of documents, such as the World Wide Web, with the purpose of "measuring" its relative importance within the set. The algorithm may be applied to any collection of entities with reciprocal quotations and references. The numerical weight that it assigns to any given element E is referred to as the PageRank of E and denoted by PR(E). Other factors like Author Rank can contribute to the importance of an entity.

A PageRank results from a mathematical algorithm based on the webgraph, created by all World Wide Web pages as nodes and hyperlinks as edges, taking into consideration authority hubs such as cnn.com or usa.gov. The rank value indicates an importance of a particular page. A hyperlink to a page counts as a vote of support. The PageRank of a page is defined recursively and depends on the number and PageRank metric of all pages that link to it ("incoming links"). A page that is linked to by many pages with high PageRank receives a high rank itself.

2 Algorithm

The original PageRank algorithm was described by Lawrence Page and Sergey Brin in several publications. It is given by

$$PR_A = \frac{1-d}{n} + d\frac{PR_{T_1}}{C_{T_1}} + \cdots + \frac{PR_{T_n}}{C_{T_n}}$$

where PR_A is the PageRank of page A,
PR_{T_i} is the PageRank of pages T_i which link to page A,
C_{T_i} is the number of outbound links on page T_i,
d is a damping factor which can be set between 0 and 1 and
n is the total number of all pages on the web.

The Random Surfer Model

In their publications, Lawrence Page and Sergey Brin give a very simple intuitive justification for the PageRank algorithm. They consider PageRank as a model of user behaviour, where a surfer clicks on links at random with no regard towards content. The random surfer visits a web page with a certain probability which derives from the page's PageRank. The probability that the random surfer clicks on one link is solely given by the number of links on that page. This is why one page's PageRank is not completely passed on to a page it links to, but is divided by the number of links on the page.
So, the probability for the random surfer reaching one page is the sum of probabilities for the random surfer following links to this page. Now, this probability is reduced by the damping factor d.The probability for the random surfer not stopping to click on links is given by the damping factor d, which is, depending on the degree of probability therefore, set between 0 and 1. The higher d is, the more likely will the random surfer keep clicking links. Since the surfer jumps to another page at random after he stopped clicking links, the probability therefore is implemented as a constant (1-d) into the algorithm. Regardless of inbound links, the probability for the random surfer jumping to a page is always (1-d), so a page has always a minimum PageRank.

2.1 Implementation

PageRank is computed iteratively;The algorithm we use to implement PageRank is the Power Iteration Method. while at the very beginning (time t) all n pages have the same score $\frac{1}{n}$, as time passes (time t + 1) the i^{th} page has instead the following PageRank value:

$$p_{i,t+1} = \frac{1-d}{n} + d\sum_{j=1}^{n} A_{ij}P_{j,t}$$

In the previous formula d represents a damping factor usually set to a value between 0 and 1 (the implementation uses d = 0:85); A_{ij} represents instead the

$(i,j)_{th}$ element of the state transition probability matrix, an incidence matrix with a stochastic structure:

$$A = \begin{bmatrix} x_{11} & & \dots & x_{1n} \\ x_{21} & \dots & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ x_{n1} & \dots & \dots & x_{nn} \end{bmatrix}$$

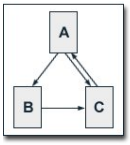
More specically the element A_{ij} represents the transition probability that the user in page i will navigate to page j, and is computed as:

$$A_{ij} = \begin{cases} \frac{1}{O_j} & if(j,i) \in E \\ \frac{1}{n} & if(j,i) \notin E and O_j = 0 \\ 0 & otherwise \end{cases}$$

Notice that O_j is the number of links - called outlinks - from page j to other pages. The termination condition is defined by the case in which two consecutive iterations of the algorithm produce two almost identical p-vectors. At the end of the algorithm we obtain the final PageRank values.

2.2 Sample Input and Output

Let's take an example to understand.



As we can see here we have a small web consisting of three pages A, B and C, whereby page A links to the pages B and C, page B links to page C and page C links to page A. According to PageRank, the damping factor d is usually set to 0.85, but to keep the calculation simple we set it to 0.5.Now we get the following equations for the PageRank calculation from formula:

$$PR(A) = 0.5 + 0.5PR(C)$$
$$PR(B) = 0.5 + 0.5(PR(A)/2)$$
$$PR(C) = 0.5 + 0.5(PR(A)/2 + PR(B)$$

Now these equations can easily be solved using Power Iteration method.Here each page is assigned an initial starting value and the PageRanks of all pages are then calculated in several computation circles based on the equations determined by the PageRank algorithm. The iterative calculation shall again be illustrated by our three-page example, whereby each page is assigned a starting PageRank value of 1.

Iteration	PR(A)	PR(B)	PR(C)
0	1	1	1
1	1	0.75	1.125
2	1.0625	0.765625	1.1484375
3	1.07421875	0.76855469	1.15283203
4	1.07641602	0.76910400	1.15365601
5	1.07682800	0.76920700	1.15381050
6	1.07690525	0.76922631	1.15383947
7	1.07691973	0.76922993	1.15384490
8	1.07692245	0.76923061	1.15384592
9	1.07692296	0.76923074	1.15384611
10	1.07692305	0.76923076	1.15384615
11	1.07692307	0.76923077	1.15384615
12	1.07692308	0.76923077	1.15384615

As we can see at last with error less then some specific value we end iterations and get our final page ranks as.

$PR(A) = 1.07692308 \qquad PR(B) = 0.76923077 \qquad PR(C) = 1.15384615$

2.3 CSR Representation of Sparse Matrix

CSR here stands for Compressed sparse row. As we are working with sparse matrix most of the elements of the matrix have 0 value. Hence we can implement special data structure for sparse matrix and we can improve both Storage an Computation time by working with only non-zero values.
The CSR format stores a sparse m x n matrix M in row form using three (one-dimensional) arrays ($val, raw - ptr, col - ind$
).

1. The array val holds all the nonzero entries of M in left-to-right top-to-bottom order.Thus of length No. of nonzero elements in matrix.

2. The array $raw - ptr$ is of length No. of raws + 1. It is defined by this recursive definition:
   IA[0] = 0
   IA[i] = IA[i−1] + (number of nonzero elements on the i^{th} row)

3. Array, $col - ind$, contains the column index in M of each element of A and hence is of length No. of nonzero element as well.

Below is an example for better understanding.

$$M = \begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

$$val = [3, 4, 5, 7, 2, 6]$$
$$col_ind = [2, 4, 2, 3, 1, 2]$$
$$raw - ptr = [0, 2, 4, 4, 6]$$

3 Parallelization

3.1 Scope of Parallelism

The PageRank algorithm, initially implemented with a adjacency matrix (directed graph), requires a time complexity of O(n^2) in each iteration, since for each element of the matrix, the new value has to be computed. Here, n is the number of nodes.

The adjacency matrix, represented by CSR representaion of sparse matrix, stores only the non-zero values, which is equal to the number of edges(e). During each iteration of power method, only the non-zero values are computed and we need to check for each node, whether any non-zero exists and hence the time complexity of the algorithm is O(n+e).

For a reallife web matrix, the values of edges are 10^{11} or higher. Thus, the algorithm is computationally expensive, and hence an attempt is made to parallelize the algorithm.

3.2 Parallelization Strategies

For parallelization of PageRank algorithm we need to first remove dependencies from algorithm.

```
curcol = 0;
for(i=0; i<n; i++){
        rowel = row_ptr[i+1] - row_ptr[i];
        for (j=0; j<rowel; j++) {
          p_new[col_ind[curcol]] += val[curcol] * p[i];
          curcol++;
        }
}
```

As we can see her in above serial code for parallel we will be writing *p_new*[] at multiple threads. Hence we need to remove it with transpose of matrix. Also instead of each time accessing p_new[] we simply declare inside variable and add multiplication each time in it.

```
#pragma omp parallel for private(j) schedule(static)
for(i=0; i<n; i++){
     float pi = 0.0;
     int rp1 = row_ptr[i+1];
     for (j=row_ptr[i]; j<rp1; j++) {
```

```
            pi = pi + val[j]*p[col_ind[j]];
        }
    p_new[i] += pi;
    }
```

For parallelization of this loop we have used #pragma omp parallel for private(j) schedule(static) syntax as we need j of inner loop private and for schedule(static) iterations blocks are mapped statically to the execution threads in a round-robin fashion.

4 Results

4.1 Speedup

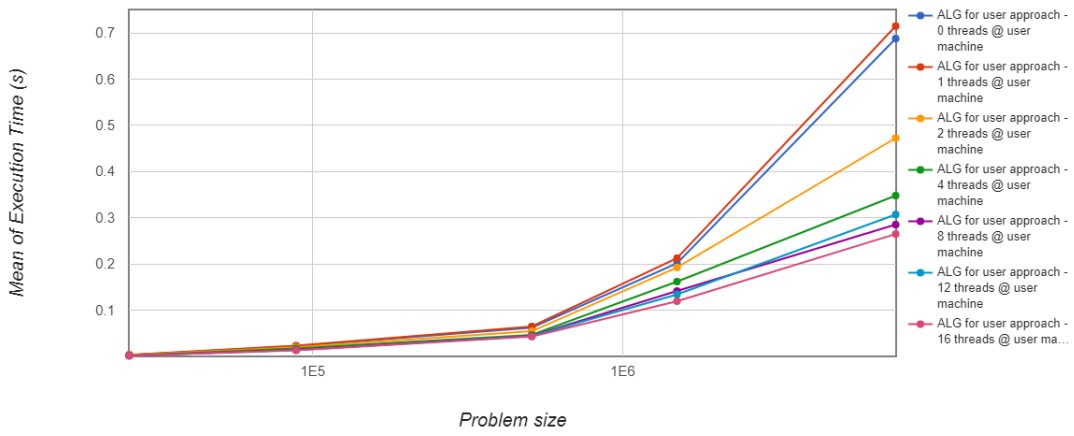


Figure 1: Problem Size Vs. Execution Time

As we can see from the figure 2 we are not getting desired speedup. Simply because the ratio of computation to memory access is extremely low here. For each iteration of the inner loop we fetch once the column index into j (4 bytes), the matrix element into data (4 bytes), the value of the vector element (8 bytes) and the previous value of the result(4 byte) (since compilers rarely optimise access to shared variables). Then you perform 2 very fast floating point operations (FLOPs) and perform a store afterwards. In total you load 20 bytes and you perform 2 FLOPs over them. This makes $\frac{1}{10}$ FLOPs per byte. Also we are calling for p[col-ind[j]] which does not fit in spatial locality well.
Thus sparse matrix-vector multiplication cannot be effectively parallelised or

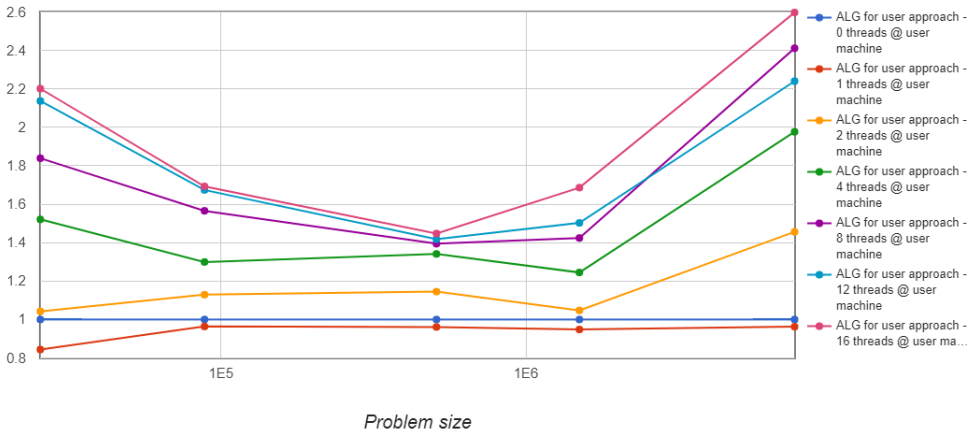


Figure 2: Problem Size Vs. Speedup

even vectorised on a single multi-core chip, unless all data could fit in the last level cache or the memory bus is really really wide.

4.2 Efficiency

As we are not getting desired speedup(figure 2), the figure 3 shows that Efficiency is decreasing with number of threads used for PageRank.

4.3 Karp-Flatt

By definition, the experimentally determined sequential fraction is a constant value that does not depend on the number of processors.

$$e = \frac{C(seq)}{T(1)}$$

On the other hand, the Karp-Flatt metric is a function of the number of processors.

$$e = \frac{\frac{1}{s} - \frac{1}{p}}{1 - \frac{1}{p}}$$

Considering that the efficiency of an application is a decreasing function on the number of processors, Karp-Flatt metric allows us to determine the importance of C(com) in that decrease.

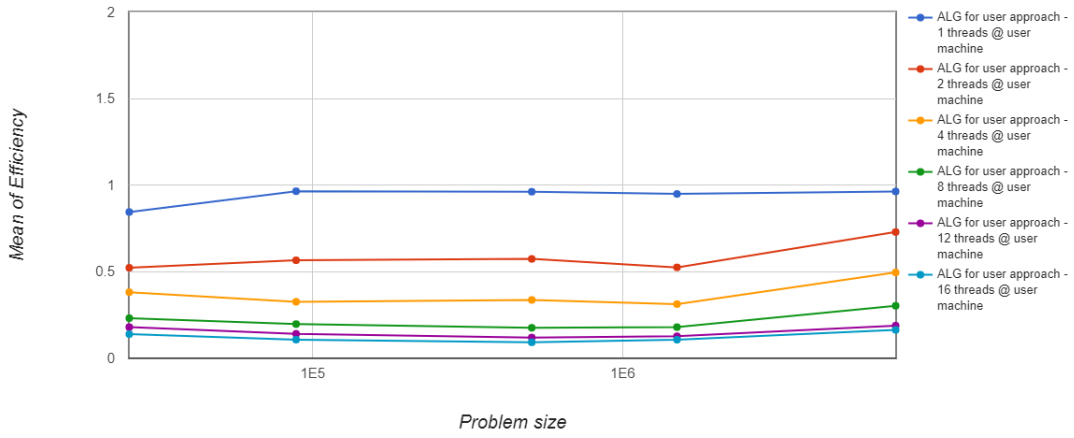


Figure 3: Problem Size Vs. Efficiency

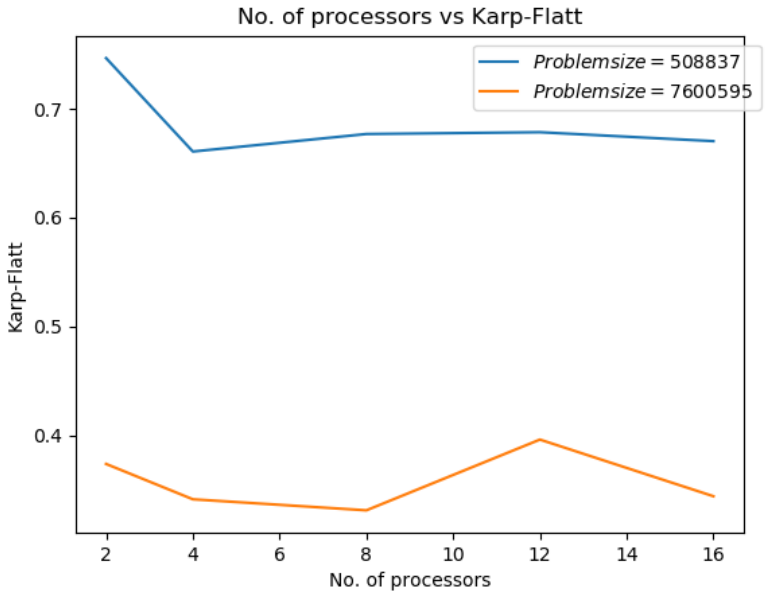


Figure 4: No. of processors Vs. Karp-Flatt

- If the values of e are constant when the number of processors increases, that means that the C(com) component is constant. Therefore, the efficiency decrease is due to the scarse parallelism available in the application.
- If the values of e increase with the increase in the number of processors, it means that the decrease is due to the C(com) component, that is, due to the excessive costs associated with the parallel computation (communication costs, synchronization and/or computation initialization).

As seen from the figure 4, the value of e generally remains constant with number of processors increasing, thus showing no extra parallel computation cost is incurred, but the parallelism is scarse. This is because, as mentioned above, more memory access are required and hence only parallelism is done for operations.

5 Future Scope for Improvement

There are simple and fast random walk-based distributed algorithms for computing PageRank of nodes in a network. They present a simple algorithm that takes $O(\log n/\epsilon)$ rounds with high probability on any web graph (n is the network size and ϵ is the reset probability) $1-\epsilon$ is called damping factor(d mentioned above) used in the PageRank computation.There are also a faster algorithm that takes $O(\sqrt{\log n}/\epsilon)$ rounds in undirected graphs. Both of the above algorithms are scalable, as each node processes and sends only small (polylogarithmic in n, the network size) number of bits per round.

6 Conclusion

We implemented pagerank algorithm with CSR representation of matrix. And although it is faster and better solution than simple matrix operations we are not able to get as good speedup as simple matrix vector multiplication.
Because of high memory access to computational operations in CSR representation PageRank(CSR matrix-vector multiplication) is defined as "memory-bound proble".

References

[1] HPC – Algorithms and Applications - Dwarf #2 – Sparse Linear Algebra. `https://www5.in.tum.de/lehre/vorlesungen/hpc/WS12/sparseLA.pdf`.

[2] Power iteration. `https://en.wikipedia.org/wiki/Power_iteration`.

[3] Stack Overflow slow slow sparse matrix vector product (CSR) using open mp. `https://stackoverflow.com/questions/10850155/whats-the-difference-between-static-and-dynamic-schedule-in-openmp`.

[4] Wikipedia PageRank. `https://en.wikipedia.org/wiki/PageRank`.