TopN Algorithm

Abstract

It is already proven by many studies that finding all pairs shortest path problem is applicable for many area including computer science and also, efficient way to study the graph.

Although we can find out the algorithm that make it which is finding all pairs shortest path more faster and efficient using parallel algorithm based on “Peng at al.’s” and “Kim at al.’s“, still it has a limitation for dealing with large real-word dataset compare to the size of computer in real word. In this paper, we suggest the new method which make it available with a reasonable amount of time and speed with the data used in real world and multi-node multi-core system.

Keywords : Parallel algorithm, shared memory algorithm, all-pairs shortest path, MPI

Introduction

The usefulness such as being used in various field of All-pairs shortest path algorithm has been proven through the lot of study dealing with it.

Previous papers proposed the newest algorithm with improved time complexity for large-scale data and more advanced methodology by combining parallel methodology and algorithm based on the first study, respectively. In this paper, we propose a new way of processing of large-scale data in real word, resulting with acceptable time and speed, adapting the preceding researches. The greatest advantage of this methodology is that multiple computers can do impossible size of job through the communication between them if there is only one computer. The data from our experiments showed mostly linear speed ups. And after experiment with few methods, we select the one which makes the program overhead minimal. The main issues will be addressed in this paper as follow: In section 2, we will briefly review the studies that have been referred to. In section 3, we will explain the implemented algorithm with how this program is developed. In section 4, we present the advantages of our program and finally in section 5, we are going to discuss about the future direction of this paper.

Parallel computation with topN algorithm

Algorithm 1

Pseudo code for the modified Dijkstra's algorithm with file method

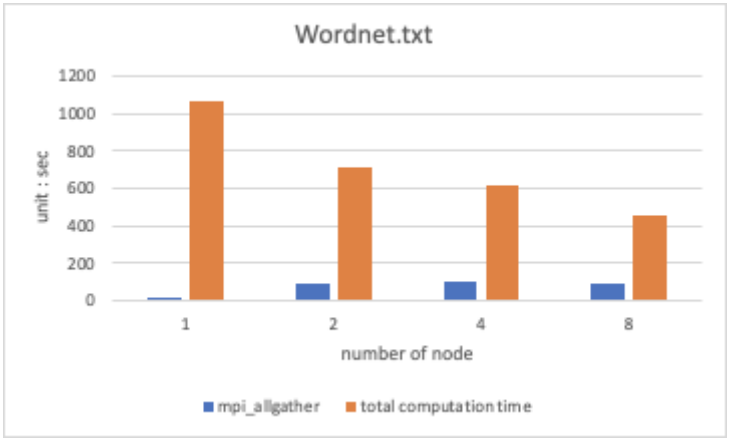
```
input: Graph G = (V , E), source s, weight matrix L,
distance matrix D, vector flag, number of total vertex N,
vector file, topN matrix topN
    i=0
    for each vertex d ∈ D do
       D[i]=0
    end for
    D[s] = 0
    Q = s
    while Q is not empty do
       t = DeQueue(Q)
       if flag[t] = 1
          If file[t] = 1 then
             coveredD = covered list of distance written in file
t
             for each vertex v ∈ V do
                if D[s,t] + coveredD[t,v] < D[s,v] then
                   D[s,v] = D[s,t] + coveredD [t,v]
                end if
             end for
          else
             for each vertex v ∈ V do
                if D[s,t] + topN[t,v] < D[s,v] then
                   D[s,v] = D[s,t] + topN [t,v]
                end if
             end for
       else
          for each edge (t,v) outgoing from t do
             if D[s,t] + L[t,v] < D[s,v] then
                D[s,v] = D[s,t] + L[t,v]
                Enqueue(Q,v)
             end if
          end for
    end while
    output: updated distance matrix D
```

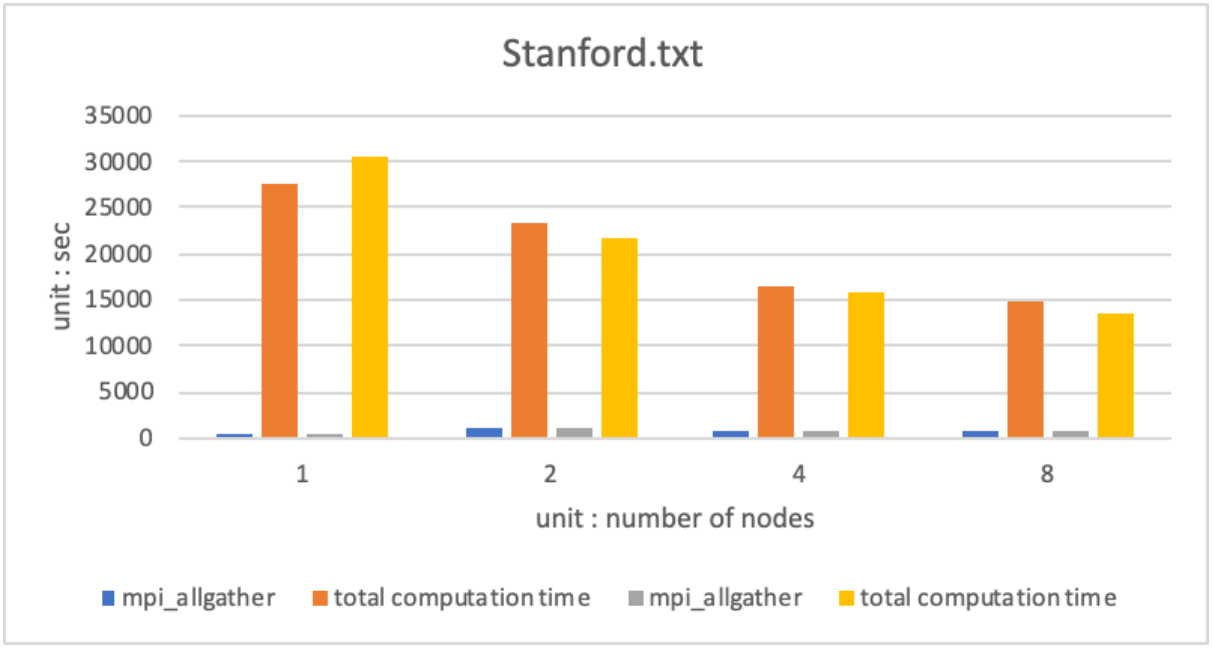
The usefulness and accuracy of reusing the result of already calculated APSP is discussed and proved by previous papers. We used the way of merging parallel program that getting a profit from fact that high-degree vertices are mainly used when calculate dijkstra. Only the difference is that the way of preserve vertex to be stored in memory or as file is decided based on the order of precedence about degree. The reason storing the high-degree based on the ordering information is for the purpose of making this design able to deal with large size of data bigger than actual size of computer and shortening the time.We use OpenMP and openMPI API for our parallel APSP algorithm under shared-memory parallel environments.

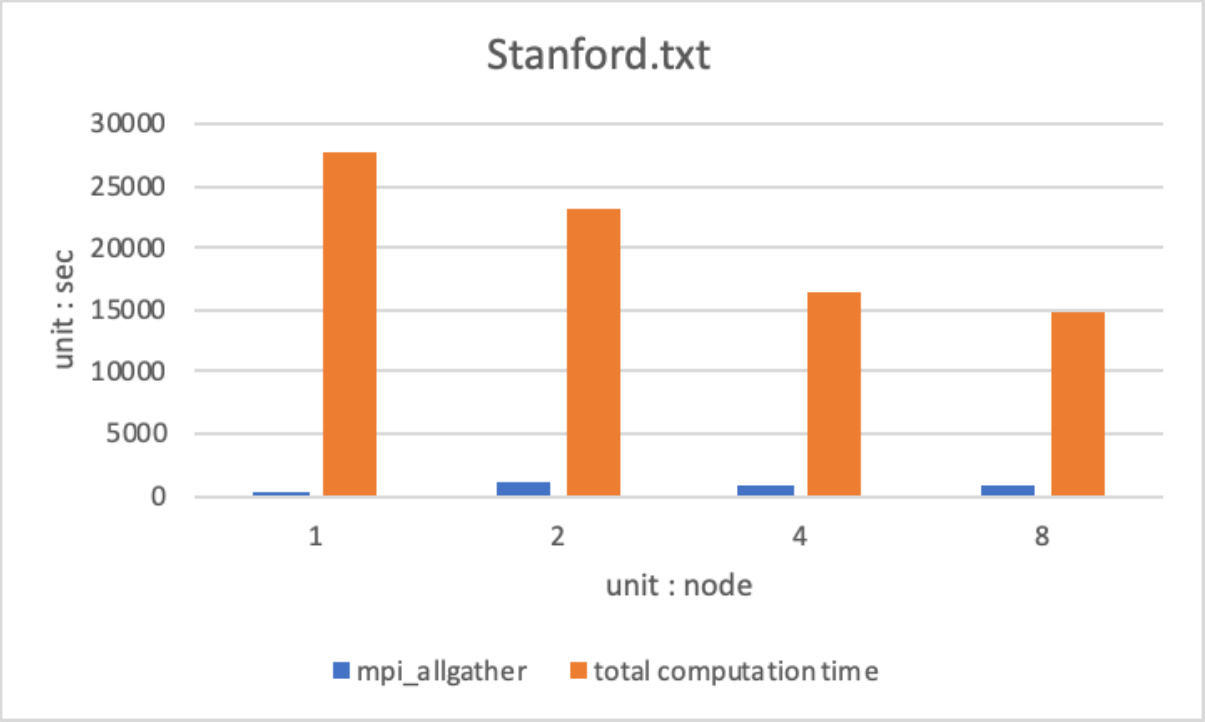
Algorithm 2
Basic algorithm for APSP Problem with file method

```
1.  input: Graph G = (V , E) and Weight Matrix L
2.
3.  Calculate the number of topN m will be stored in memory
4.
5.  Do initialization
6.     for each vertex (v) do
7.        D[v]= ∞
8.     end for
9.     for i = 1 to m do
10.       topN[i]=0
11.    end for
12.    for i = 1 to n do
13.       flag[i] = 0
14.       file[i]=0
15.    end for
16.
17. find the descending sorted order of vertices w.r.t. the degrees by the MultiLists
    ordering procedure
18. Find max/min degree of the given graph.
19. Initialize the multiple lists of buckets for all of the threads.
20.
21. Calculate all-pairs shortest path
22.    for i = 1 to n do
23.       If i<m then
24.          call the modified Dijkstra's procedure (Algorithm 1) using the source of
    index order[i]
25.          topN=D;
26.          Copy upaded topN data and deliver to all nodes
27.       Else
28.          call the modified Dijkstra's procedure (Algorithm 1) using the source of
    index order[i]
29.          Save each result as file to disk
30.       End if
31.    end for
32. output: distance matrix D and files
```

When we implement a parallel APSP solution based on Peng et al.'s and EAPSP algorithm, there is two ways. One way is to read a file in algorithm1 and the other on is calculates the dijkstra without reading the file in algorithm3.







Algorithm 1 which makes real-word data set calculation available by adopting file read and write has a similar structure with “Kim et al.’s” final algorithm. The most big difference with “Kim et al.’s” is we are added a file method at loop and dijkstra scope where run the SSSP code multiple times. Also, calculation for estimating the size of memory will allocated for TopN and variable named “file “ which tell running vertex is using file method or not is added.

For the loop scope repeating SSSP, an array containing the ordering information based on the high degree density is used. Since the starategy to less the burden to input data is store the precalculated only number of data named as topN at memory, only the topN order is calculated and stored in memory.

For the accurate result of calculation, all nodes are using openMPI’s function mpi_allgather for communication between them since each node only has result of calculated distance infromation of allocatated ones. It causing a synchronization problem and solved with openMP’s pragma omp barrier,pragma omp single.

After all vertices the amount of topN finishing, rest of vertices are sotred as file at disk. In the case of file method, communications between nodes are not required except threads.

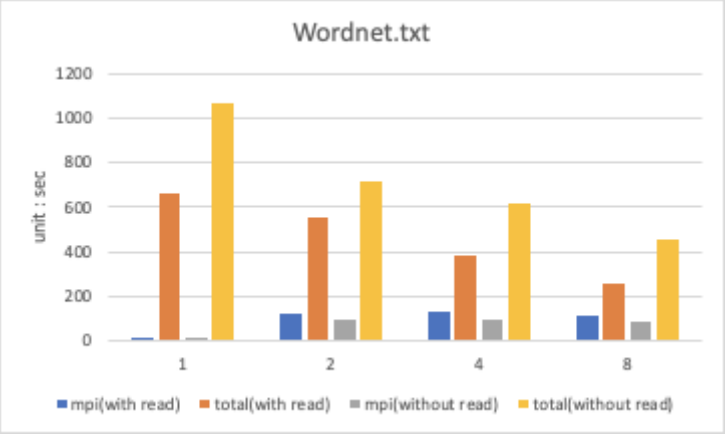
File overhead problem

Algorithm 3
Pseudo code for the modified Dijkstra's algorithm without file method

```
input: Graph G = (V , E), source s, weight matrix L,
distance matrix D, vector flag, number of total vertex N,
topN matrix topN
i=0
for each vertex d ∈ D do
   D[i]=0
end for
D[s] = 0
Q = s
while Q is not empty do
   t = DeQueue(Q)
   if flag[t] = 1
         for each vertex v ∈ V do
            if D[s,t] + topN[t,v] < D[s,v] then
               D[s,v] = D[s,t] + topN [t,v]
            end if
         end for
   else
      for each edge (t,v) outgoing from t do
         if D[s,t] + L[t,v] < D[s,v] then
            D[s,v] = D[s,t] + L[t,v]
            Enqueue(Q,v)
         end if
      end for
end while
output: updated distance matrix D
```

We can find out the total compuation time is lot more taken compare to EAPSP. There are two consumption about this result. First one is use of static-cyclic scheduling instead of dynamic-cyclic scheduling and second is excessive time consumping on file method. Considering the necessery of giving up dynamic-cyclic scheduling because of design which using node and thread as source of distruibution vertices between them, we made a brief experiment of file method by editing our original design.

By deleting file read part from calculating dijkstra part and make it using only infromation from topN it self, we can earn result on the right bottom.



1	2	4	8
16.6258272	116.626672	125.249405	110.576733
660.170919	554.824049	386.310281	252.322564
12.4992425	89.618528	95.2357813	88.595794
1065.1128	713.604914	613.511402	450.745621