International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 4– No.8, December 2012 – www.ijais.org

Some Computational Results on MPI Parallel Implementation of Derived Subgraph Algorithm

E. M. Badr
Scientific Computing Department, Faculty of Computer Science and Informatics, Benha University, Benha, Egypt.

ABSTRACT

The aim of this paper is to present an experimental evaluation of a parallel derived subgraph algorithm *PDSA* using *MPI*. The performance of the algorithm *PDSA* is verified by computational experiments on some special graphs with different size, run in a cluster of workstations. *MPI* seems to be appropriate for these kind of experiments as the results are reliable and efficient.

Keywords

Union closed sets conjecture, induced graphs, derived subgraphs, parallel algorithms, parallel processing.

1. INTRODUCTION

A union-closed family of sets A is a finite collection of sets not all empty such that the union of any two members of A is also a member of A . The following Conjecture is due to Peter Frankl [1, 2, 3].

Conjecture 1. Let $A = \{ A_1, A_2, \ldots, A_n \}$ be a union-closed family of n distinct sets. Then there exists an element which belongs to at least $n / 2$ of the sets in A.

Let $A = \cup A_i$. If we replace each set A_i by $B_i = A - A_i$ then we get an intersection-closed family of sets, which we call the dual family of A. Therefore Conjecture 1 is equivalent to the following.

Conjecture 2. Let $B = \{ B_1, B_2, \ldots, B_n \}$ be an intersection-closed family of n distinct sets. Then there exists an element which belongs to at most $n / 2$ of the sets in B .

An induced subgraph S of a graph G is called a derived subgraph of G if S contains no isolated vertices. An edge e of G is said to be residual if e occurs in more than half of the derived subgraphs of G otherwise e is non-residual. Let $D(G)$ denote the set of derived subgraphs of G and put $n_d(G) = |D(G)|$. A graph-theoretic version of the union-closed sets conjecture due to El-Zahar [4]. He formulated a weaker version of Conjecture 1 specialized for graphs as the following.

Conjecture 3. Every non-empty graph contains a non-residual edge.

B. Llano et al proved that the every simple graph with at least one edge contains a non-residual edge (Conjecture 3) [5].

In this paper we examine the computational performance of a parallel version of the sequential derived subgraph algorithm *SDSA* [6] on some special graphs with different size. We perform experiments using the communication package Message Passing Interface (MPI) [7]-[8]-[9] . Our preliminary results reveal that if the parallel version of the derived subgraph algorithm run in 16 processors then we can achieve a speed-up factor of about 3.5 times faster comparing with the sequential version of the derived subgraph algorithm.

The paper is organized as follows: In Section 2 we introduce a serial derived subgraphs for a given graph G. The parallel derived subgraphs algorithm *PDSA* is presented in Section 3. To continue with, some preliminary computational results on some special graphs are reported in Section 4. Finally, we present the conclusion in Section 5.

2. Serial Derived Subgraph Algorithm

In this Section, we introduce a serial derived subgraphs algorithm SDSA [6] which calculates the number of derived subgraphs for a given graph G . The algorithm also determines the residual and non-residual edges. The parameters of the algorithm are :

$A[i, j]$: the adjacency matrix of G.

$S[i]$: all of the subsets of $V(G)$.

(i,j) : the entry of the matrix $E(i,j)$ which is equal to the number of derived subgraphs that contain $v_i v_j$

total : the number of all derived subgraphs of G.

Let G be a graph which has n vertices and m edges. We can represent the graph G by the Adjacency-Graph class, where $a[i][j]$ is the entry element (i,j) in the adjacency matrix A. The algorithm finds all subsets of the vertex set $V(G)$; then it checks if the current subset induces a derived subgraph or not. The algorithim finds the number of derived subgraphs that contain any edge $e \in E(G)$.

Our main algorithm *SDSA* calls three procedures Initialize-Subset, Get-Next-Subset and Check-Subset as follows:

Algorithm 1: A serial derived subgraphs algorithm *SDSA*

Input : $A[i][j]$ the adjacency matrix of G.

Output : (total) the number of all derived subgraphs of G.

```
1: Call Algorithm2 ( Initialize-Subset )
2: while Not Done do
3:        Call Algorithm3 ( Get-Next-Subset )
4:        Call Algorithm4 ( Check-Subset )
5:        if DERIVED then
6:                total ← total + 1
7:                for i=1 → n do
8:                for j=i+1 → n do
9:                        if S[i] = S[j] = 1 then
10:                               E[i,j] ← E[i,j] + 1
11:                       end if
12:               end for
13:       end for
14:       end if
15: end while
16: Return total
17: For each e =(v_i, v_j) if E[i,j] > total / 2 the edge e is residual otherwise is non-residual.
```

The Initialize-Subset procedure initializes the initial subset of $V(G)$ as array $S[j] = 0$. The subgraph induced by the initial S is the empty derived subgraph. We outline below the initialize-Subset procedure which considers the empty subgraph as the first derived one.

Algorithm 2: Initialize-Subset

```
1: Take the empty set to be the initial subset S
2: Set the value of total = 1
3: For every edge e = ( i, j ) let E[i,j] = 0
4: Done ← False
```

The Get-Next-Subset procedure generates all subsets of $V(G)$ by the method is known as a binary counting representation.

Algorithm 3: Get-Next-Subset

```
1: j ← n + 1
2: repeat
3:        j ← j – 1
4: until (( S[j] = 0) or ( j = 0)
5: if j ≠ 0 then
6:        S[j] ← 1
7:        MAX ← j
8:        for i= MAX +1 → n do
9:                S[i] = 0
10:       end for
11: else
12:       Done ← True
13: end if
```

The Check-Subset verifies the current subset S as a derived subgraph or not. A precise description of this process is the following.

Algorithm 4: Check-Subset

```
1: DERIVED ← False
2: count ← 1
3: for k = 1 → n do
4:        if S[k] = 1 then
5:                sum = 0
6:                for j = 1 → n do
7:                    sum = sum + a[k][j] *S[j]
8 :                   if sum ≠ 0 then
9:                        sum ← 1
10:                       count ← count * sum
11:                   end if
12:               end for
13:       end if
14: end for
15: if count ≠ 0 then
16: DERIVED ← True
17: end if
```

3. MPI Parallel Version of the Derived Subgraph Algorithm.

We introduce a parallel derived subgraphs algorithms *PDSA* which cluclates the number of derived subgraphs for a given graph *G*. The main idea in this algorithm is that each processor generates $2^n/NPRS$ subsets such that *NPRS* is the number of processors and *n* is the size of a adjacency matrix of a graph *G*.

Parallel Derived Subgraph Algorithm

```
Begin
1-  /* All processors read the adjacency matrix A[ ] [ ] */
    for 1 ≤ i ≤ n do
            for 1 ≤ j ≤ do
                Read A[i][j]
2-  /* All processors initialize the set S[ ] and the matrix E[ ] [ ] */
    for 1 ≤ i ≤ n do
            Set S[i]=0
    for 1 ≤ j ≤ n do
            for 1 ≤ k ≤ n do
                    Set E[j][k]=0
3-  /* for 2^n subsets each processor generates q = 2^n/NPRS subsets */
    for  0 ≤ i ≤ NPRS  pardo
            Each processor generates q subsets only.
4-  /* Each processor check their subsets */
    Set Derived = 0
    Set COUNT = 1
    for 1 ≤ k ≤ n do
    Begin
            if S[k] = 1
            begin
                    Set SUM = 0
                    for 1 ≤ k ≤ n do
                        Set SUM = SUM + A[k][j]*S[j]
                    If ( SUM<>0
                            Set SUM = 1
                  Set COUNT = COUNT * SUM
               End
     End
    If ( COUNT <> 0)
       Set DERIVED = 1
    Else
       Set DERIVED = 0
    If (DERIVED = 1 )
    Begin
       Set mytotal = mytotal + 1
       for 1 ≤ i ≤ n do
        for i+1 ≤ j ≤ n do
            if ( S[i] = 1 and S[j] = 1 )
                    Set E[i][j] = E[i][j] + 1
    End
5-  Combine the values " mytotal " from all processors and put it in "total".
6-  /* Each processors checks all the edges are residual or not */
    Set myresidual = 0
    for 1 ≤ i ≤ n do
    Begin
            for i+1 ≤j≤ n do
            Begin
            if ( A[i][j] = 0 and E[i][j] > total/2)
                    Set myresidual = myresidual + 1
            End
    End
7-  Reduce the values " myresidual" on all processors to a single value " residual".
```

4 Computational Results

The algorithm described in Section 3 has been experimentally implemented. In this Section, the numerical experiments are presented. It must be mentioned that the computational results demonstrate a speedup for the *PDSA* algorithm on some special graphs (path graph, the cyclic graph, complete graph and bipartite graph).

All test runs were carried out on 16 uniprocessors Intel Pentium III 500MHz with 512 KB L2 Cache. The processors were interconnected using Fast Ethernet and Scalable Coherent Interface (SCI). Furthermore, the machine precision was 32 decimal digit. The reported CPU times were measured in seconds. MPI implementation MPICH v.1.2.6 was used and appropriately configured for our cluster. Usage of this machine was provided by the National University of Athens, School of Electrical and Computer Engineering.

TABLE 1

P_{18} **No_DS** = 1 7991 & **No_NRE** = 17		
No. Processors	***Time (secs.)***	***Speed up***
1	4.318319	1.0000
2	3.299953	1.3086
4	2.823169	1.5296
8	2.439094	1.7705
16	1.290226	3.3469

TABLE 2

P_{20} **No_DS** = 55405 & **No_NRE** = 19		
No. Processors	***Time (secs.)***	***Speed up***
1	20.275985	1.0000
2	14.582843	1.3904
4	13.490342	1.5030
8	11.733871	1.7280
16	6.151799	3.2959

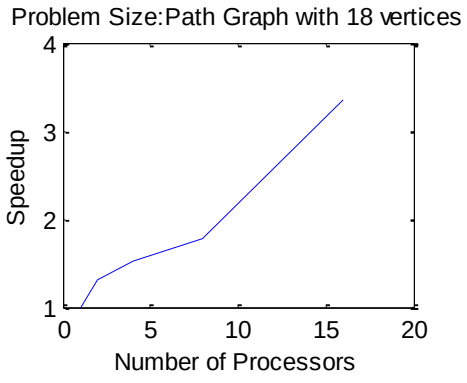

Fig.1 *PDSA* on a path graph with 18 vertices

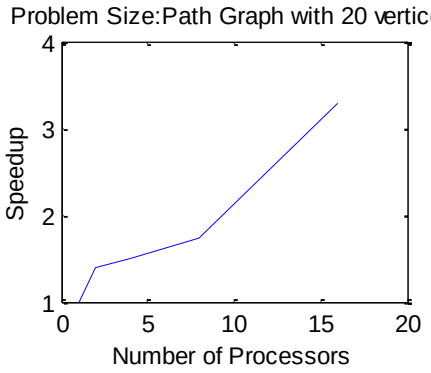

Fig.2 *PDSA* on a path graph with 20 vertices

Figure 1 to 8 show the computation performance of various problem sizes over various numbers of CPUs. The problem sizes are the path graph with 18 and 20 vertices, the cyclic graph with 18 and 20 vertices, the complete graph with 18 and 20 vertices and the complete bipartite graph with 18 and 20 vertices. Each data set was executed 3 times so we reported the average time.

TABLE 3

C_{18} **No_DS** = 24914 & **No_NRE** = 18		
No. Processors	***Time (secs.)***	***Speed up***
1	4.139703	1.0000
2	3.275599	1.2638
4	2.790309	1.4836
8	2.489729	1.6627
16	1.303223	3.1765

TABLE 4

C_{20} **No_DS** = 76725 & **No_NRE** = 20		
No. Processors	***Time (secs.)***	***Speed up***
1	20.632824	1.0000
2	14.860864	1.3884
4	13.642449	1.5124
8	11.845963	1.7418
16	6.209620	3.3227

Someone may ask, why then not using smaller problem sizes? Well the answer to this, is that if we use the smaller problem size, we can not get a good speed up because the communication time is greater than the computation time.

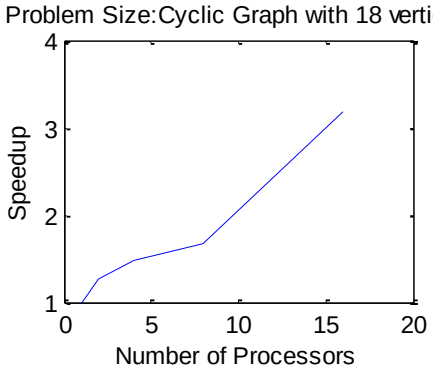

Fig.3 *PDSA* on a cycle with 18 vertices

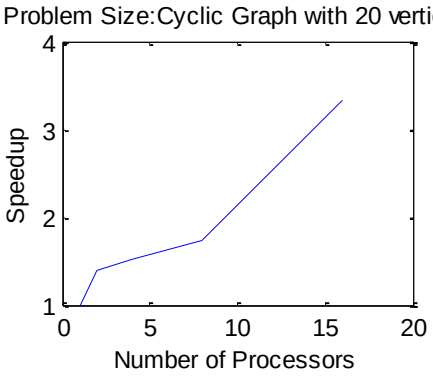

Fig.4 *PDSA* on a cycle with 20 vertices

First of all it is well understandable that for 1 CPU, as a separate source code was developed, communication time does not exist, because the master does not send anything to any worker. It does all the computation on its own. It had to be done using this pattern, for having accurate results and the reason was that the same computer architecture should be used. If the code for excution on 1 CPU was executed on machine with a newer CPU the times should be quite different. As a result this should be executed on the head of the cluster called as " master".

TABLE 5

K_{18} ***No_DS*** = 262126 & ***No_NRE*** = 153		
No. Processors	***Time (secs.)***	***Speed up***
1	4.968775	1.0000
2	3.822134	1.3000
4	3.294506	1.5082
8	2.864973	1.7343
16	1.518253	3.2727

TABLE 6

K_{20} ***No_DS*** = 1048556 & ***No_NRE*** = 190		
No. Processors	***Time (secs.)***	***Speed up***
1	24.708195	1.0000
2	17.700548	1.3959
4	16.143871	1.5305
8	13.893427	1.7784
16	7.212622	3.4257

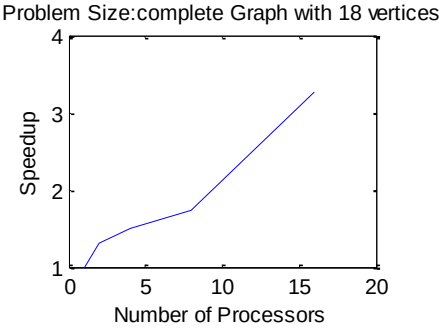

Fig.5 *PDSA* on K_{18}

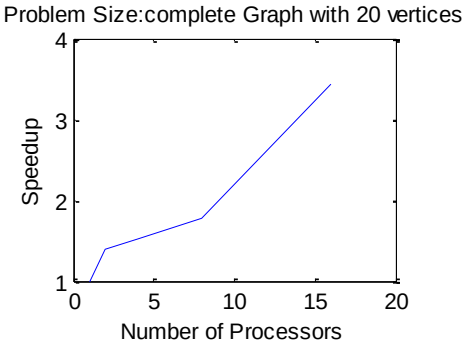

Fig.6 *PDSA* on K_{20}

Looking at Table 8, comparing the time for 1 CPU and the time for 16 CPUs, we can understand the difference in time, and how many times faster the results occur with parallelization. To be more specific with 16 CPUs and the complete bipartite graph with 20 vertices, we can achieve a speed up of 3.4 times ($time_{cpu[1]} / time_{cpu[16]}$).

TABLE 7

$K_{9,9}$ **No_DS** = 261122 & **No_NRE** = 81		
No. Processors	***Time (secs.)***	***Speed up***
1	4.987248	1.0000
2	3.590531	1.3890
4	3.296264	1.5130
8	2.863984	1.7414
16	1.509900	3.3030

TABLE 8

$K_{10,10}$ **No_DS** = 1046530 & **No_NRE** = 100		
No. Processors	***Time (secs.)***	***Speed up***
1	24.832957	1.0000
2	17.753043	1.3988
4	16.174660	1.5353
8	13.834818	1.7950
16	7.245751	3.4272

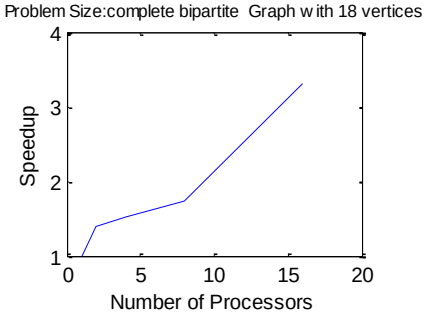

Fig.5 *PDSA* on $K_{9,9}$

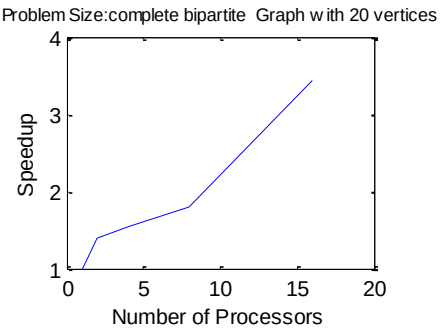

Fig.6 *PDSA* on $K_{10,10}$

5. Conclusion

We have presented a parallel implementation of the derived subgraphs algorithms *PDSA*. The proposed implementation has an advantage. It leads to important reduction in total solution time of a derived sungraphs problem. The performance analysis also, shows that the speed up obtained is highly sensitive to communication among the processors.

6. References

[1] I. Rival (Ed.), *Graphs And Order*, Reidel, Dordrecht-Boston,(1985), p.25.

[2] R. P. Stanley, *Enumerative Combinatorics*, vol. I, Wadsworth & Broks/Cole, Belmont, CA, (1986).

[3] B. Poonen, *Union-Closed Families*, J. Combin. Theory, A 59 (1992), 253-268.

[4] M. H. El-Zahar , A Graph-Theoretic Version Of The Union-Closed Sets Conjecture, J.Graph Theory 26 (1997), no. 3, 155-163.

[5] B. Llano, J. Montellano-Ballesteros, E. Rivera-Campo and R. Strauz " On Conjecture of Frankl and El-Zahar" J. Graph Theory 57: 344-352 (2008).

[6] M. I. Moussa and E. M. Badr, A Computational Study for the Graph-Theoretic Version of the Union-Closed Sets Conjecture, International Journal of Computer Applications, Volume 50 – No.12, July 2012.

[7] I. Foster, Desiging and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. Addison-Wesley, 1995.

[8] W.Groop, E. Lusk, and A. Skjellum, Using MPI: Portable Parallel Programming with the Message Passing-Interface. MIT Press, 1994.

[9] E. M. Badr, M. I. Moussa, K. Paparrizos, N. Samaras and A. Sifaleras, Some Computational results on MPI Parallel Implementations of Dense Simplex Methods, Transactions on Engineering, Computing and Technology, vol. 17, pp. 228-231, 2006.