About parallelization of an algorithm for the maximum clique problem

Iliya Bouyukliev, Venelin Monev,
Maria Dzhumalieva-Stoeva

Albena, 2013

Introduction

Def. ***Finite undirected graph*** is an ordered pair G = (V, E), where:

- $V = \{v_1, v_2, \ldots, v_n\}$ is finite set of vertices
- $E = \{e_1, e_2, \ldots, e_m\}$ is finite set of undirected edges and each element $e_k \in E$

$(k = 1, 2, \ldots, m)$ is unordered pair $\{v_i, v_j\}$, $v_i, v_j \in V, 1 \le i, j \le n$.

Def. Let G = (V, E) be a graph and $V' \subseteq V$. If we remove from E every edges $\{v_i, v_j\}$, such that $v_i \not\in V'$ or $v_j \not\in V'$, but the others remain, then $G'(V',E')$ is called a **subgraph** of G.

Introduction

Def. Let G = (V, E) be a graph, such that $\{v_i, v_j\} \in E$, $for\ \forall v_i, v_j \in V, i \neq j$. Then G is called **complete**.

Def. Let G'=(V',E') be a complete subgraph of $G = (V, E)$, $V' \subseteq V$. Then G is called clique or m-clique, if $|V'|$ = m. A **maximum** clique is a clique of the largest possible size m in a graph G.

// maximal clique is a clique that cannot be enlarged //

Some clique problems

- finding the maximum clique (with the largest number of vertices);
- solving the decision problem of testing whether a graph contains a clique larger than a given size;
- // finding all maximal cliques.

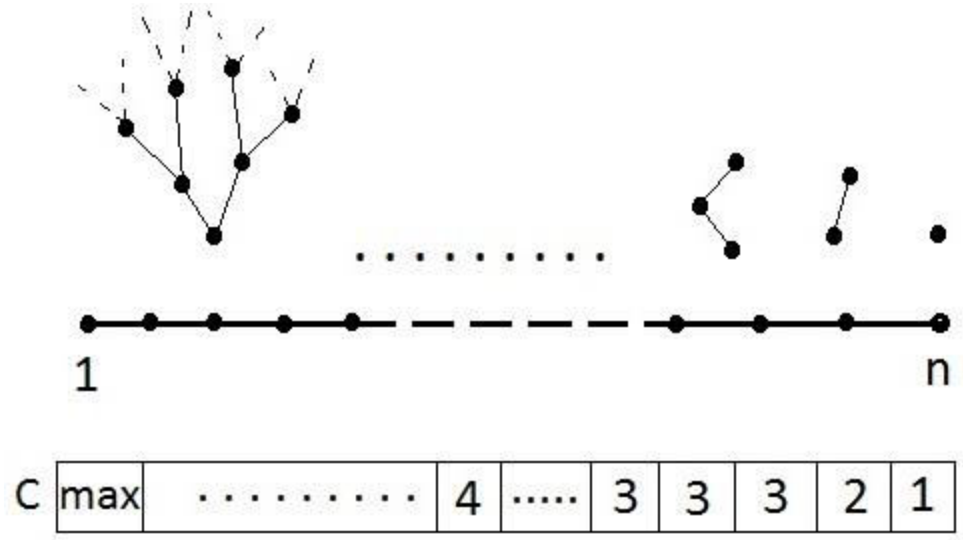
We choose one of the most efficient algorithms for finding maximum clique [4-Cliquer] and trying to implement it in parallel.

Cliquer – description

- G(V, E) , |V| = n
- V = $\{v_1, v_2, \ldots, v_n\}$
- $\boldsymbol{Si} = \{v_i, v_{i+1}, \ldots, v_n\}$
- $\boldsymbol{N(v_i)}$ – set of vertices adjacent to a vertex v_i.
- **c[*n*]** – array contains largest clique in $\boldsymbol{Si}$ for v_i

```
Func {
        max : = 0
        for  $i$ : = n downto 1
        {    found : = false
             clique ($Si \cap N(v_i)$,  1)
             c[$i$] : = max
        }
  }
```

```
clique (U, tsize){
        if  ( |U| = 0  && tsize > max)
        {       max : = tsize
                /save the solution/
                return
        }
        while(U $\neq 0$ ){
                if (tsize + |U| $\leq$ max) { return }
                $i$ : = min { $j$ | $vj \in$ U}
                if (tsize + c[$i$] $\leq$ max) { return }
                U : = U \ {$vi$}
                clique (U $\cap$ $N(vi)$, size + 1)
                if (found = true) {return}
        }
}
```



Let v_i is contained in **tsize** - clique and

$N(v_i) = \{v_{i1}, \ldots, \; v_{ij}, \ldots, \; v_{i\kappa}\}$

--> We search for **m** - clique

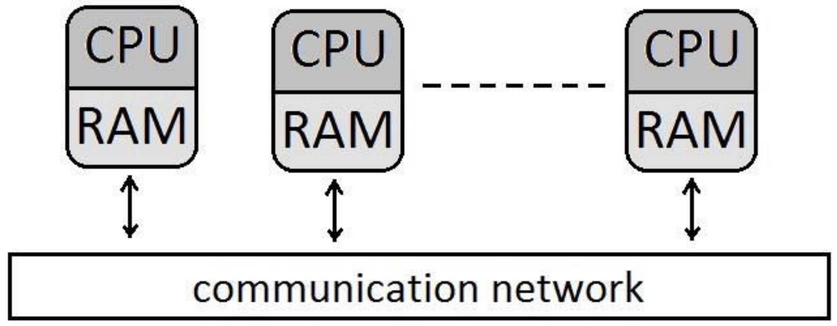
if (tsize + c[ij] $\leq$ m) --> true

--> v_{ij} is not proper.

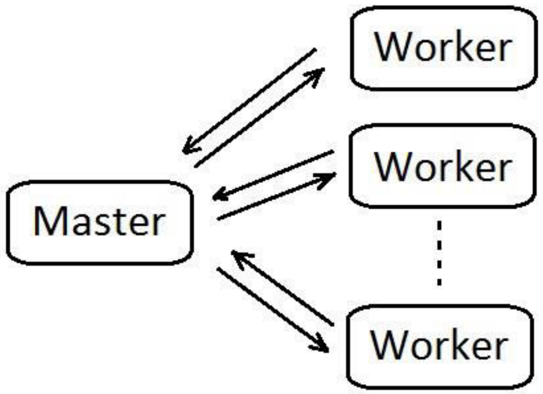
Parallel implementation

- Using MPI library (Message Passing Interface) with C.
- SPMD (Single Program Multiple Data) model.
- Master/worker strategy.
- Blocking or nonblocking communication.

MPI

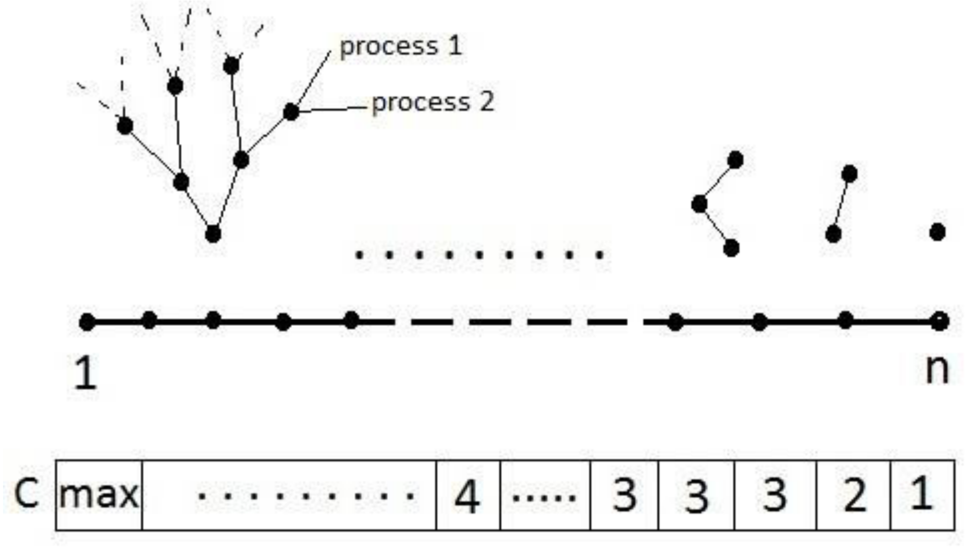


Master/worker strategy



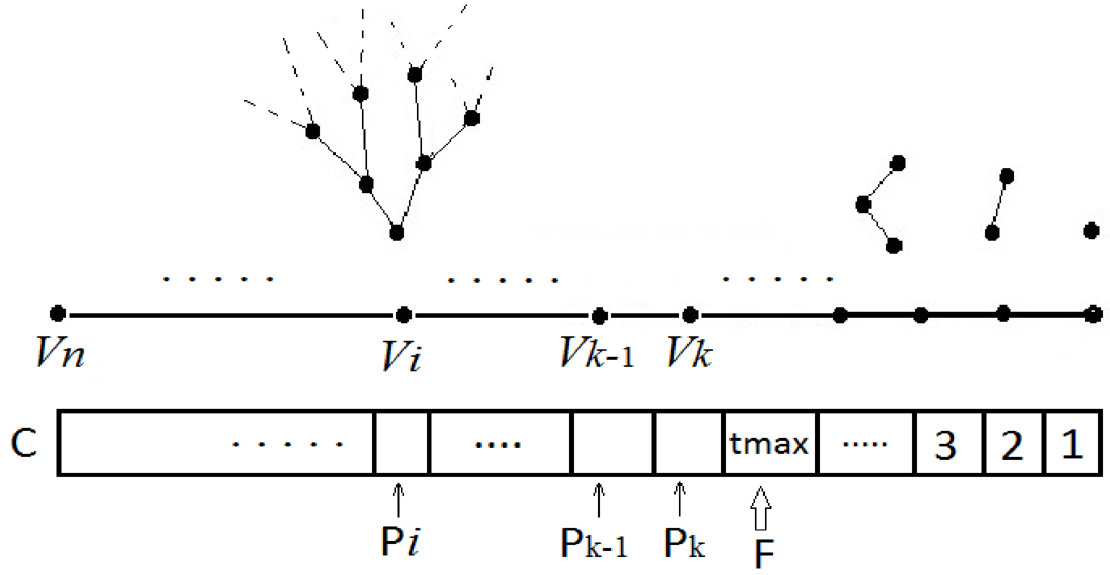
Known implementations

•Single process works on fixed branch of the searching tree [7].



Our implementation

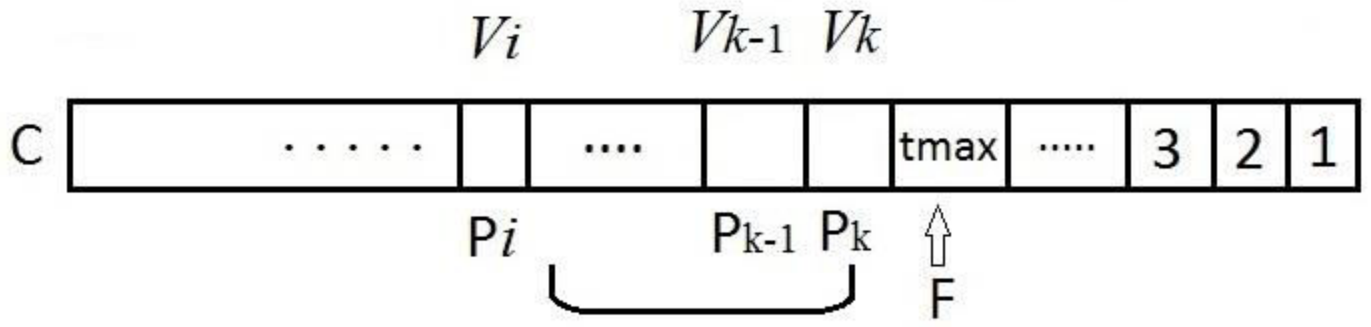
•Each next vertex of a graph is given to separate process Pi (All processes operate simultaneously for different vertices with corresponding search trees).



tmax – the size of current maximum clique.
F – position of the last vertex (that work has been completed).
P*i* – worker process responsible to find a clique in S_i (which contains v_i) with size greater than **maxl**, but not larger than **maxu**, keeps this value in **maxl** and sends it to the master .

Cliquer searches a clique with size plus 1 than the current, and if the next joined vertex does not participate in bigger clique, it will be not proper.

In our case we are searching for maximum clique with size *m'*, where
maxl < *m'* ≤ **maxu**. We use two bounds **maxl** and **maxu**, because the processes before **P*i*** are still working and their results are undefined.



Worker statement

U : array of sets
found: boolean
size, i , maxl, maxu: integer
maxl - lower bound, maxu – upper bound
C[n], Si, $N(vi)$ – the same like in a Cliquer algorithm

```
Mainclique(i, maxl, maxu)
 {
            found:= false;
            $U$[0] := {$Si \cap N(vi)$};
            cliquep(1, maxl, maxu);
 }
```

```
cliquep (size, maxl, maxu)
{
    if |$U$[size − 1]| = 0 then
    {
        if (size > maxl) then
        {
            maxl:=size;
            if (maxl = maxu) then found:=true;
        }
    }
    while |$U$[size − 1]| <> 0 do
    {
        if (size + |$U$[size − 1]| $\leq$ maxl) then return;
        $i$ := $min\{ j : vj \in U$ [size − 1]};
        if (( size + $C$ [$i$] ) $\leq$ maxl) then return;
        $U$[size − 1] := $U$[size − 1] \ {$vi$};
        $U$[size] := { $U$[size − 1] $\cap$ $N$($vi$) };
        cliquep (size+1, maxl, maxu);
        if (found = true) then return;
    }
}
```

- Any worker receives values of i, $maxl$, $maxu$ and information to improve the local array C from the master.
- Master sends values of i, $maxl$, $maxu$, C and receives the obtained value of $maxl$ from any worker.

<u>There are two versions of this implementation:</u>

***** Master waits for a response from the process which has received his work first. Here the assumption is that the work on the vertex with smaller index will be heavier in the most of the cases. (blocking communication)

****** Master gives work to the first ready process. (nonblocking communication)

```
Clique_paralel* (G(V,E));
{
    if (master)
    {      Send initial work for all workers;
           work := |V|- numworkers;
           while (work>0) do
            {     wait_message_from_next;
                   receive_from_next (maxl,workdone,workernum);
                   update C;
                   find "maxl"’ and "maxu" for current "work";
                   send_to_workernum(maxl, maxu, work, info for C);
                   work:=work-1;
           }
     }
     if (worker)
     {while (work>0) do
        {
             receive_from_master(maxl, maxu, work, info for C);
             update C;
             Mainclique(work, maxl, maxu);
             send_to_master(maxl, workdone, workernum);
        }
     }
}
```

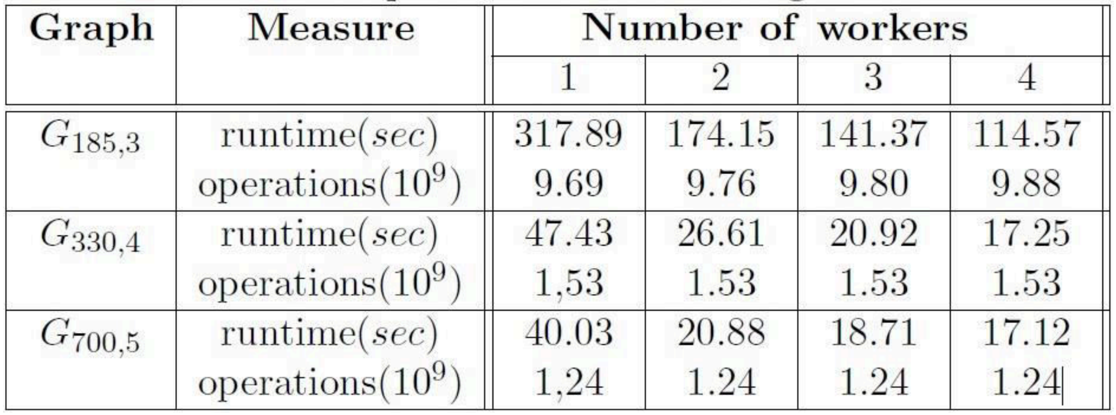
```
Clique_paralel** (G(V,E));
{
    if (master)
    {       Send initial work for all workers;
            work := |V|- numworkers;
            while (work>0) do
             {
                    receive_from_any (maxl,workdone,workernum);
                    update C;
                    find "maxl"’ and "maxu" for current "work";
                    send_to_workernum(maxl, maxu, work, info for C);
                    work:=work-1;
            }
     }
     if (worker)
     {while (work>0) do
        {
              receive_from_master(maxl, maxu, work, info for C);
              update C;
              Mainclique(work, maxl, maxu);
              send_to_master(maxl, workdone, workernum);
        }
     }
}
```

Experimental results

We use graphs $\boldsymbol{G}_{p,q}$ with the following property: The vertex set is $V = \{v_1, v_2, \ldots, v_p\}$. There is an edge from v_i to v_j if and only if the Hamming distance between the binary representations of i and j is larger than $q - 1$.

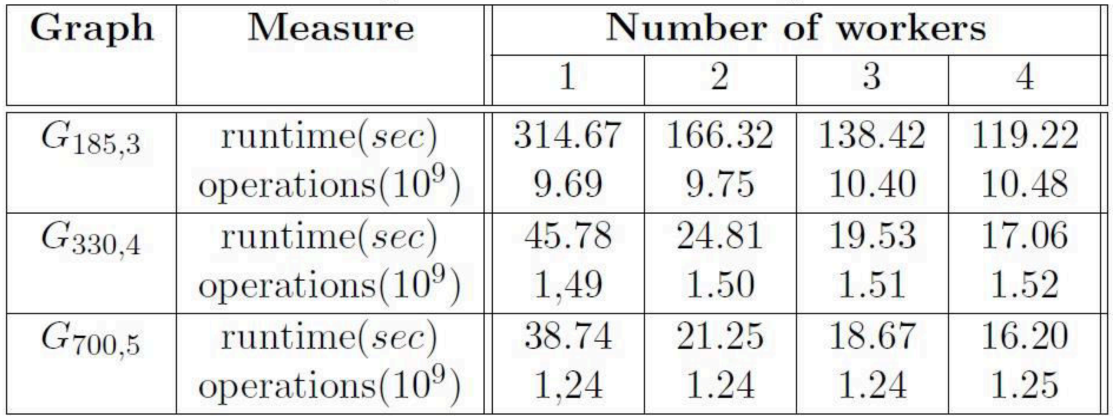
Experimental results

Table 1: Computational results of Algorithm 1



Graph	Measure	Number of workers			
		1	2	3	4
$G_{185,3}$	runtime(sec)	317.89	174.15	141.37	114.57
	operations(10^9)	9.69	9.76	9.80	9.88
$G_{330,4}$	runtime(sec)	47.43	26.61	20.92	17.25
	operations(10^9)	1,53	1.53	1.53	1.53
$G_{700,5}$	runtime(sec)	40.03	20.88	18.71	17.12
	operations(10^9)	1,24	1.24	1.24	1.24

Table 2: Computational results of Algorithm 2



Graph	Measure	Number of workers			
		1	2	3	4
$G_{185,3}$	runtime(sec)	314.67	166.32	138.42	119.22
	operations(10^9)	9.69	9.75	10.40	10.48
$G_{330,4}$	runtime(sec)	45.78	24.81	19.53	17.06
	operations(10^9)	1,49	1.50	1.51	1.52
$G_{700,5}$	runtime(sec)	38.74	21.25	18.67	16.20
	operations(10^9)	1,24	1.24	1.24	1.25

References

[1] I. Bouyukliev and E. Jakobsson, Results on binary linear codes with minimum distance 8 and 10, *IEEE Trans. Inform. Theory*, **57**, 6089–6093, 2011.

[2] S.A. Cook, The Complexity of Theorem Proving Procedures, in *STOC ’71: Proceedings of the third annual ACM symposium on Theory of computing*, Shaker Heights, Ohio, United States, 1971, 151– 158.

[3] W. Gropp, E. Lusk and A. Skjellum, *UsingMPI: Portable parallel programming with the message-passing Interface*, The MIT Press, 1994.

[4] P.R. J. Östergård, A fast algorithm for the maximum clique problem, *Discrete Applied Mathematics*, **120**, 197–207, 2002.

[5] CP.R.J. Östergård, Constructing combinatorial objects via cliques, in: *Surveys in Combinatorics*, B. S. Webb (Editor), Cambridge University Press, Cambridge, 57–82, 2005.

[6] P. R. J. Östergård, T. Baicheva and E. Kolev, Optimal binary One-Error-Correcting codes of length 10 have 72 codewords, *IEEE Trans. Inform. Theory*, **45**, 1229–1231, 1999.

[7] Chad Brewbaker, Parallel implementation of the CLIQUER algorithm for the computing the MAX CLIQUE of a random graph of order 900, 2005.

[8] P. S. Pacheco, *Parallel programming with MPI*, San Francisco, Calif.: Morgan Kaufmann Publishers, 1997.