Seventh International Workshop on Optimal Codes and Related Topics
September 6-12, 2013, Albena, Bulgaria *pp. 53-58*

About parallelization of an algorithm for the maximum clique problem[1]

Iliya Bouyukliev iliya@moi.math.bas.bg
Institute of Mathematics and Informatics, Bulgarian Academy of Sciences
P.O.Box 323, 5000 Veliko Tarnovo, Bulgaria
Venelin Monev venelinmonev@gmail.com
Faculty of Mathematics and Informatics, Veliko Tarnovo University,
5000 Veliko Tarnovo, Bulgaria
Maria Dzhumalieva-Stoeva mdzhumalieva@gmail.com
Institute of Mathematics and Informatics, Bulgarian Academy of Sciences
P.O.Box 323, 5000 Veliko Tarnovo, Bulgaria

Dedicated to the memory of Professor Stefan Dodunekov

Abstract. Many construction and classification problems in coding theory can be considered as a maximum clique problem. In this paper we consider some approaches for parallel implementation of an algorithm which solves this problem.

1 Introduction

A *graph* is an ordered pair $G = (V, E)$ which consists of a vertex set V and a collection E of 2-element subsets of V. Each element of E is called an edge. A *hypergraph* $H = (V_H, E_H)$ consists of a vertex set V_H and a collection E_H of subsets of V_H. Each element of E_H is called a hyperedge.

A *clique* in a graph G is a subset of the vertex set $Cl \subset V$, such that for every two vertices in Cl, there exists an edge connecting them. A maximum clique is a clique of the largest possible size in a given graph. An *independent set* is a set of vertices in a graph, no two of which are adjacent. A maximum independent set is a largest independent set for a given graph. Every clique corresponds to an independent set in the complement graph.

Many construction and classification problems in coding theory can be considered as a maximum clique problem [5] [6] or the problem for maximum independent set in the complement graph.

The construction of linear codes can be considered as a maximum independent set problem in a hypergraph [1]. An independent set in H is a subset S of V_H that does not contain any edge. A maximum independent set in a given hypergraph H is an independent set of maximum cardinality. Since these are NP-hard problems, no polynomial time algorithms are expected to be found.

[1]This research is partially supported by the Bulgarian NSF under Contract I01/0003/2012

Nevertheless, many instances of this problems can be solved with realizations similar to the algorithm in [4]. In this paper we consider some approaches for parallel implementation of this algorithm. Different ways for development of parallel algorithms for solving a maximum clique problem are presented in [7]. They are related to parallelization of the back-track search.

2 Main ideas and notations in the serial algorithm

In this section we give the notations and main ideas of an algorithm for finding maximum clique presented by Östergård [4].

Consider the graph $G(V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$. The set of vertices adjacent to a vertex v is denoted by $N(v)$ and the number of vertices in the graph is n. The variable max, which is global, gives the size of a maximum clique when the algorithm terminates. Let $S_i = \{v_i, v_{i+1}, \ldots, v_n\}$ and the function $c(i)$ gives the largest clique in S_i. The values of the function $c(i)$ are saved in the array C. The strategy of the algorithm is to consider cliques in S_n that contain v_n, then cliques in S_{n-1} that contain v_{n-1}, etc. Suppose that the algorithm finds $l = C[i]$ in the $(n - i)$th step (l is a maximum clique in S_i). In the next step the algorithm will search for a clique with size $l + 1$ which contains the vertex v_{i-1}. The possible values for $C[i - 1]$ are l or $l + 1$. In the search process the algorithm uses already obtained $C[i], C[i + 1], \ldots, C[n]$ for the pruning strategy in the search tree (see line 14 of Algorithm 2 in [4]). If in some step the algorithm has found a clique with size $l + 1$ the variable $found$ becomes $true$, the algorithm stops to search and goes to the next step.

3 Parallel implementations - notation and pseudocode

For the implementation of the algorithms we use the MPI interface [8], [3]. MPI (Message Passing Interface) is a standardized programming environment for distributed-memory parallel computers. The basic concept of MPI is message passing: one process sends data via message to another process. If exactly two processes are involved in this situation, it is called point to point communication. There are two kind of communications: blocking and nonblocking. In the blocking form the process waits until the communication has completed. Otherwise, in the nonblocking communication a process does not block and may check for this completion while perform some other task in meantime. The programming model SPMD (Single Program Multiple Data) is commonly used.This is when many processes are all running the same program and they will be working on different parts of the program's data. Many combinatorial problems (like current) used backtracking algorithms for their solving. Backtrack search is a potential example for parallelization and the well-known master/worker pattern is more preferred. This patterns consists of two logical nodes: a master

and one or more instances of a worker. The master sets up specific work to the workers and manages their performance.

Any worker process is 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. The worker do this by Algorithm 1. Any worker receives values of i, $maxl$, $maxu$ and information to improve the local array C from the master. The master only distributes the work. It sends values of i, $maxl$, $maxu$ and information for improvements of the array C and receives the obtained value of $maxl$ from any worker.

We present two versions of this implementation. In the first one the master waits for a response from the process which has received his work first. This is written by lines with (*) in the pseudo code of Algorithm 2. This version is based on the assumption that the work of Algorithm 1 in the vertex with smaller index will be heavier in most of the cases.

In the second version the master gives work to the first ready process. This is written by a line with (**) in the pseudo code of Algorithm 2.

The first version is implemented by blocking communication, because the master process waits to receive messages back from the worker processes in strict order. This is not the case in the second version. It implements nonblocking communications and each result from a worker will be send to the others by the master.

We have made experimental results on the dual core four tasks INTEL i3 processor. We use graphs $G_{p,q}$ 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 $q - 1$.

The calculational times and approximate number of operations for the two versions are given in Table 1 and Table 2.

Remark 1: This implementation is proper only for sparse graphs.

Remark 2: It is known that many problems are reducible to clique problem. As an example satisfiability (or SAT) [2], which is a fundamental problem in computer science, since solutions to many important problems, such as circuit testing, combinatorial search, cryptography algorithms, and so forth, can be formulated as satisfiability questions.

References

[1] I. Bouyukliev, 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.

Algorithm 1.

```
|   var U: array of sets;
|         found:boolean;

Procedure cliquep( size, var maxl, var maxu:integer);
| var i:integer;
|  {
|    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]| <= maxl) then return;
|        i := min{j : v_j ∈ U[size − 1]};
|        if ((size + C[i]) <= maxl) then return;
|        U[size − 1] := U[size − 1]\{v_i};
|        U[size] := {U[size − 1] ⋂ N(v_i)};
|        cliquep(size+1,maxl,maxu);
|        if found = true then return;
|      };
| };

Procedure Mainclique(i, var maxl,var maxu:integer);
| {
|       found:=false;
|       U[0] := {S_i ⋂ N(v_i) };
|       cliquep(1,maxl,maxu);
|  };
```

Algorithm 2.

```
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);
        ** 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);

    }
  }
}
```

Table 1: Computational results of Algorithm 1

Graph	**Measure**	**Number of processors**			
		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

[3] W. Gropp, E. Lusk, 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] P. 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, E. Kolev, Optimal binary One-Error-Correcting codes of length 10 have 72 codewords, *IEEE Trans. Inform. Theory,* **45**, 1229–1231, 1999.

[7] C. 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.