Design and Implementation of Efficient Parallel Algorithm for 3-SAT Problem

A Dissertation submitted to the University of Hyderabad in partial fulfillment of the

degree of

MASTER OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE

by

ANKA RAO NOMULA

DEPARTMENT OF COMPUTER & INFORMATION SCIENCES

SCHOOL OF MATHEMATICS & COMPUTER / INFORMATION SCIENCES

University of Hyderabad
(P.O.) Central University, Gachibowli
Hyderabad – 500 046
Andhra Pradesh
India

CERTIFICATE

This is to certify that the dissertation entitled "**Design and Implementation of Efficient Parallel Algorithm for 3-SAT Problem**" submitted by **Anka Rao Nomula** bearing Reg. No **09MCMI22** in partial fulfillment of the requirements for the award of **Master of Technology** in **ARTIFICIAL INTELLIGENCE** is a bonafide work carried out by him under my supervision and guidance.

The dissertation has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Prof. Narendra S. Chaudhari
Project Supervisor,
Department of CSE,
IIT Indore.

Dr. Rajeev Wankar
Project Supervisor,
Department of CIS,
University of Hyderabad.

Head,
Department of CIS,
University of Hyderabad.

Dean,
School of MCIS,
University of Hyderabad.

DECLARATION

I, **Anka Rao Nomula** hereby declare that this Dissertation entitled "**Design and Implementation of Efficient Parallel Algorithm for 3-SAT Problem**" submitted by me under the guidance and supervision of **Prof. Narendra S. Chaudhari** and **Dr. Rajeev Wankar** is a bonafide work. I also declare that it has not been submitted previously in part or in full to this University or other University or Institution for the award of any degree or diploma.

Date: **ANKA RAO NOMULA**

09MCMI22

Acknowledgement

Apart from the efforts of me, the success of my project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project.

I would like to express my greatest gratitude to my project supervisor ***Prof. Narendra S. Chaudhari****, for his valuable and continuous support throughout the progress of the project. During his visits as a visiting Professor, he spared lot of time to explain his algorithm and made me understand the logic in depth. His suggestions helped me to improve my work a lot.*

I would like to express my greatest gratitude to my project supervisor ***Dr. Rajeev Wankar****, for his valuable and continuous support throughout the progress of the project. His suggestions helped me to improve my work a lot.*

I am extremely thankful to ***Prof. Arun Agarwal,*** *for providing us both freedom and an excellent lab facility where we could test our work effectively.*

I wish to thank my parents for their undivided support and interest who inspired me and encouraged me to go my own way, without whom I would be unable to complete my project.

At last but not the least I want to thank my friends who appreciated me for my work and motivated me.

Finally, I would like to thank God, who made all the things possible.

ANKA RAO NOMULA

Abstract

Polynomial algorithm for 3-SAT problem has been reported in [1, 2]. As a first step, we have developed a sequential implementation in C++ for Polynomial algorithm for 3-SAT. To speed up computations, we identified inherent parallelism in the portion of algorithm.

We have proposed PRAM based parallel algorithm for the polynomial algorithm of 3-SAT problem with parallel time complexity of O (n^{10}) with O (n^3) number of processors. Subsequently, we tested parallel algorithm on parallel machine with multi cores. We have used Open-MP compiler directives to implement this parallel version. The sequential implementation and parallel implementation are tested upon standard datasets [5] and results are reported.

Key words: PRAM model of computation, Open-MP, P vs. NP.

Table of content

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Brief Description of New formulation for 3-SAT algorithm

3-SAT is the well known problem of computer science and has been studied by many researchers for a long time. There are several attempts that have been made to find polynomial solution for this problem but all of them have failed.

3-SAT is about the discovery of truth-value assignments of variables that result in the satisfaction of all clauses. To model this high level description of the problem, represent the 3-Clause as a conditional having a literal pair as antecedent, and single literal as consequent [1].

[1] Involves (the formulation of) two main concepts. First is the construction of logical consequents of all antecedent literal-pairs as our analysis progresses. Second is the demonstration that, while the *forward* propagation of the global effect is captured in our suitably formulated conditional, the (corresponding) *backward* propagation of the global effect is captured by enforcing the satisfaction of the contra positive of the concerned conditional.

Let F be 3-SAT formula. It consists of 3-clauses. A 3-clause has an equivalent representation with a literal-pair as an antecedent and the remaining literal as a consequent, *i.e.* $(p \vee q \vee r) \equiv (\bar{p} \wedge \bar{q}) \rightarrow r \; (\equiv (\bar{q} \wedge \bar{p}) \rightarrow r)$

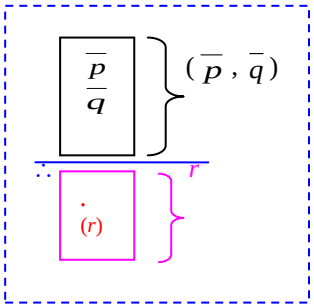

Fig1. A representation of 3-clause ($p \vee q \vee r$): consisting of (i) antecedent literal-pair $(\bar{p}, \bar{q})$ and (ii) the consequent (r).

For a clause $(p \vee q \vee r)$, we have six antecedent literal-pair representations; they are:

(1) $(\bar{p} \wedge \bar{q}) \rightarrow r$, (2) $(\bar{p} \wedge \bar{r}) \rightarrow q$, (3) $(\bar{q} \wedge \bar{r}) \rightarrow p$,

(4) $(\bar{q} \wedge \bar{p}) \rightarrow r$, (5) $(\bar{r} \wedge \bar{p}) \rightarrow q$, and, (6) $(\bar{r} \wedge \bar{q}) \rightarrow p$.

In two dimensional array $C_{a, b}$ (indices a, b being the literals), we store the clause consequents.

The 3-SAT problem requires us to satisfy a clause, say *C*, in the context of other (satisfied) clauses. This constraint may result in the restriction of the truth-value assignment(s). This restriction is formulated as $C \rightarrow s$. The well-known inference rule, Modus Ponens (MP), reminds us that $C \wedge (C \rightarrow s) \Rightarrow s$ [1].

- Algorithmic Formulation
  - Algorithm: W-Closure(A)
  - Algorithm: 3-SAT-Satisfiability(F)
  - Algorithm: Truth-Value Assignment(F)
- Complexity
  - Algorithm: W-Closure(A) : O (n^5)
  - Algorithm: 3-SAT-Satisfiability(F) : O (n^{13}) {=O (n^5) x O (n^6) x O (n^2) }

1.2 Description of Work Done

1.2.1 We have modified 3-SAT-Satisfiability (F) algorithm [1] to include step 0. Step 0 discusses mainly two properties.

- Row analysis to discover the stable literals
- Symmetry check and symmetry establishment, if needed

1.2.2 Implemented the sequential algorithm for 3-SAT problem [1], tested on datasets [5] and results are reported.

1.2.3 Designed the PRAM based parallel algorithm for 3-SAT problem [1].

- Algorithmic Formulation
  - PRAM Algorithm: W-Closure(A)
  - PRAM Algorithm: 3-SAT-Satisfiability(F)
  - PRAM Algorithm: Truth-Value Assignment(F)
- Complexity
  - PRAM Algorithm: W-Closure (A): O (n^4) time complexity with O (n^2) number of processors.
  - PRAM Algorithm: 3-SAT-Satisfiability (F): O (n^{10}) {=O (n^4) x O (n^6))} with O (n^3) number of processors.

1.2.4 Implemented the PRAM based parallel algorithm for 3-SAT problem, tested on datasets [4] and results are reported.

1.3 Organization of Thesis

A sequential 3-SAT algorithm is discussed in detail in chapter 2 and sequential time complexity is arrived. Chapter 3 includes the timing requirement for obtaining the results of the different datasets in sequential implementation. To speed up the process, we converted the algorithm in Chapter 2 to include some parallelism. This PRAM parallel algorithm and its time complexity are given in Chapter 4. Chapter 5 includes experimental test bed. Chapter 6 includes the timing requirement for obtaining the results of different datasets in parallel implementation. We have included sample datasets and results of datasets as appendices in this report.

3-SAT algorithm is about to tell whether the given formula is satisfiable or not. If given formula is satisfiable, this algorithm is able to give the truth value assignments for all the variables [1]. We have also implemented this truth value assignment algorithm along with the 3-SAT algorithm. The truth value assignments for some of the datasets are included as appendices in this report.

Chapter 2

Sequential Algorithm for 3-SAT

2.1 Sequential Algorithm

Polynomial Algorithm for 3-SAT problem is reported in [1]. For the purpose of our implementation, we have modified 3-SAT algorithm [1] to include step 0.

Algorithm: *W*-Closure (*A*);

1. Initialize $B = \Phi$; Satisfy-Flag = **true**;
2. Repeat steps 2.1 to 2.5 below (until there is no update in *B*) {
   - 2.1 *b*-update = **false**; $R = A \cup B$; // working set of restricted literals
   - 2.2 Repeat steps 2.2.1 to 2.2.3 below (until there is no change in *R*) {
     - 2.2.1 if there exists a literal as well as its negation in *R*, then { Satisfy-Flag = **false**; exit *W*-Closure; }
     - 2.2.2 else (*i.e.*, if there does not exist a literal as well as its negation in *R*) { $B = B \cup W_{m,l}$, for $m, l \in R$; // .. update *B, R* to include the literals *n* such that $R = R \cup B$; } // .. $n \in W_{m,l}$, for $m, l \in R$
     - 2.2.3 } until there is no change in *R*;
   - 2.3 For all literals *c* (*c* in { $c_1, c_2, \ldots, c_p$ }, p = 2*number of variables not in *R*) we initialize the *p* lists, $L_{c1}, L_{c2}, \ldots, L_{cp}$, as follows: $L_{ci} = \{c_i\} \cup \{n,$ a literal of variable not in $R \mid n \in W_{x,ci},$ for some $x \in R\}$ $(i = 1, \ldots, p)$;
   - 2.4 Repeat steps 2.4.1 to 2.4.4 below (until there is no change in the lists L_{ci} $(i = 1, \ldots, p)$) {
     - 2.4.1 For all lists L_c, for all *l* in L_c, include $\bar{c}$ in $L_{\bar{l}}$; // .. include contra positives
     - 2.4.2 For all lists L_c, update it (i.e., L_c) to include the literals *n* (of variables not in *R*), such that $n \in W_{m,l}$, for some $m, l \in R \cup L_c$;
     - 2.4.3 if there exists *k*, such that L_k contains a literal as well as its negation, then { update *B, R* to include $\bar{k}$; *b*-update = **true** };
     - 2.4.4 } until there is no change in the lists L_{ci} $(i = 1, \ldots, p)$;
   - 2.5 } until ((not *b*-update));
3. Return *Satisfy-Flag, B, R,* and p (= 2*number of variables not in *R*) lists L_{ci} $(1 \le i \le p)$.

Algorithm: 3-SAT-Satisfiability (F);

I. For all literal-pairs (a, b) in $L \times L$, let $C_{a,b}$ denote the list of clause consequents of the antecedent literal pair (a, b). Initialize $W_{a,b}$ to $C_{a,b} \cup \{a, b\}$. In this initialization, we note that $W_{a,a} = \{a\}$. For all literal pairs (a, b), initialize $F_{a,b}$ to truth-value **true**.

II. Declare (and initialize) global variables needed for *W*-Closure( ). Thus, declare and initialize *B*, set of literals, to null-set Φ. Initialize *Satisfy-Flag* to **true**. For all literals *a* in *L* , initialize lists L_a to null-set Φ.

III. Let *T* be list of (top level) restricted literals; initialize *T* to null-set Φ. Initialize *Satisfiable* = **true**.

IV. Repeat steps 0, ... ,5 (until there is no change) { *Change* = **false**;

0.a. For all literals x in L { // ... row analysis to discover stable literals
0.a.1. For all literals y in L − {x} {
0.a.2. if $(W_{x,y} \cap W_{x,\bar{y}}) \subset W_{x,x}$ then {
a.2.1. $W_{x,x} = W_{x,x} \cup (W_{x,y} \cap W_{x,\bar{y}})$;
a.2.2. Change = **true**;
a.2.3. }; … end of if in step 0.a.2
0.a.3. }; // … end of For loop in step 0.a.1
0.a.4. }; // … end of For loop in step 0.a

0.b. For all literal pairs (x, y) in L ×L with x < y, {
// ... symmetry check and symmetry establishment, if needed
0.b.1. if $W_{x,y} \neq W_{y,x}$ then {
0.b.2. $W_{x,y} = W_{x,y} \cup W_{y,x}$;
0.b.3. $W_{y,xy} = W_{x,y}$;
0.b.4. Change = **true**;
0.b.5. }; // … end of if in 0.c.1
0.b.6. }; // … end of For loop in step 0.c

1. For all literal pairs (x, y) in $L \times L$, not in *T* , { // ... literal-pair analysis
1.1. initialize (the lists, or sets) $W_{x,y} = T \cup W_{x,x} \cup W_{y,y} \cup W_{x,y}$;
1.2. invoke *W*-Closure($W_{x,y}$); // .. results in *Satisfy-Flag*,
// .. and *B*, set of restricted literals
1.3. if (*Satisfy-Flag*) then $W_{x,y} = W_{x,y} \cup B$ // .. include restricted literals
1.4. else {
1.4.1. $F_{x,y}$ = **false**; // .. $x \wedge y$ = **false** $\Leftrightarrow \overline{x \wedge y}$ = **true**;
1.4.2. if $\bar{y} \notin W_{x,x}$ then { $W_{x,x} = W_{x,x} \cup \{\bar{y}\}$; *Change* = **true**; }
1.4.3. if $\bar{x} \notin W_{y,y}$ then { $W_{y,y} = W_{y,y} \cup \{\bar{x}\}$; *Change* = **true**; }
1.4.4. }; // .. end of else in step 1.4
1.5. }; // .. end of For loop in step 1

2. For all literals *x* in *L* , { // .. Top level restricted literals
2.1. if $\bar{x} \notin T$ and $W_{x,x}$ includes a literal as well as its negation {
2.2. $F_{x,x}$ = **false**;
2.3. $T = T \cup \{\bar{x}\}$;
2.4. invoke *W*-Closure(*T*); // .. results in *Satisfy-Flag*, and *B*, set of literals

2.5. $T = T \cup B$; // .. include restricted literals
2.6. if ((not *Satisfy-Flag*)) then {
2.6.1. *Satisfiable* = **false**;
2.6.2. Exit For loop in step 2 & loop in step IV
2.6.3. }; // .. end of if of step 2.6
2.7. *Change* = **true**; }; // .. end of if in step 2.1
2.8. } ; // .. end of For loop in step 2

3. For all literals x in L , { // .. Diagonal Contra positive closure
3.1. if (($\bar{x} \notin W_{\bar{a},\bar{a}}$) and ($a \in W_{x,x}$)) then {
3.2. $W_{\bar{a},\bar{a}} = W_{\bar{a},\bar{a}} \cup \{\bar{x}\}$;
3.3. invoke *W*-Closure($W_{\bar{a},\bar{a}}$); // .. results in *Satisfy-Flag*, and *B* literals
3.4. if (*Satisfy-Flag*) then $W_{\bar{a},\bar{a}} = W_{\bar{a},\bar{a}} \cup B$; // .. include restricted literals
3.5. if ((not *Satisfy-Flag*)) then {
3.5.1. $F_{\bar{a},\bar{a}}$ = **false**;
3.5.2. $T = T \cup \{a\}$;
3.5.3. invoke *W*-Closure(*T*); // .. results in *Satisfy-Flag*, and *B*
3.5.4. $T = T \cup B$;
3.5.5. if ((not *Satisfy-Flag*)) then {
3.5.5.1. *Satisfiable* = **false**;
3.5.5.2. Exit For loop in step 3 & loop in step IV
3.5.5.3. }; // .. end of if in step 3.5.4
3.6. }; // .. end of if of step 3.5
3.7. *Change* = **true**; }; // .. end of if in step 3.1
3.8. } ; // .. end of For loop in step 3

4. For all literals x in L , { // .. Diagonal Transitive closure
4.1 For all literals $a \in W_{x,x}$ such that $W_{a,a} \not\subset W_{x,x}$ {
4.2 $W_{x,x} = W_{x,x} \cup W_{a,a}$; // .. transitivity closure for the case $W_{a,a} \not\subset W_{x,x}$
4.3 invoke *W*-Closure($W_{x,x}$); // .. results in *Satisfy-Flag*, and *B*, set of literals
4.4 if (*Satisfy-Flag*) then $W_{x,x} = W_{x,x} \cup B$; // .. include restricted literals
4.5 if ((not *Satisfy-Flag*)) then {
4.5.1 $F_{x,x}$ = **false**;
4.5.2 if $\bar{x} \notin T$ then {
4.5.3 $T = T \cup \{\bar{x}\}$;
4.5.4 invoke *W*-Closure(*T*); // .. results in *Satisfy-Flag*, and *B*
4.5.5 $T = T \cup B$;
4.5.6 if ((not *Satisfy-Flag*)) then {
4.5.6.1 *Satisfiable* = **false**;
4.5.6.2 Exit For loop in step 4 & loop in step IV
4.5.6.3 }; // .. end of if in step 4.5.6
4.5.7 }; // .. end of if of 4.5

4.6 *Change* = **true**; }; // .. end of For loop in step 4.1
4.7 } ; // .. end of For loop in step 4

5. Until (not *Change*)); // .. end of IV

V. If *Satisfiable* then *T* contains list of top-level restricted literals, $W_{x,x}$ contains set of restricted literals when x = **true** and $W_{x,y}$ contains set of restricted literals when the pair of literals (x , y) = (**true, true)**.

2.2 Description of W-Closure (A) formulation

The several features of the computations involved in W-Closure (A) are

(i) The W-Closure (A) constructs a set R of restricted literals.

(ii) During this construction, we don't allow the deletion of literals. As a consequence, the set R of restricted literals may contain a literal as well as its negation. This allows us to conclude that the collection A of literals is unsatisfiable.

(iii) One consequence of our discipline of not permitting the deletion of literals is, we always have $R \supseteq A$; in other words, R would possibly contain additional literals not in A. When R does not contain a literal as well as its negation, the set of additional literals is meaningful; let the set of these additional literals included in R be B.

(iv) During our analysis involved in W-Closure (A), we successively construct consequent lists for all variables not included in R.

(v) We terminate our analysis when we discover that there is no change in the consequent lists for all variables not included in R.

(vi) When we decide to terminate the analysis involved in W-Closure (A), we note that, for each literal of the non-restricted variable (i.e., for each variable whose literal is not included in R), we have the corresponding consequent list.

(vii) We note that, due to (vi), W-Closure (A) also returns p(=2*number of variables not in R) consequent lists (These lists are potentially useful for discovering the truth-value assignments when the given set of literals A can satisfy F).

This, we note that W-Closure(A) takes, as input, the non-empty set of literals A, and it generates: (i) *Satisfy-Flag* (**false** if our computational analysis in W-Closure(A) reveals that set of literals A cannot satisfy F), (ii) a set B of restricted literals (of some variables in the set of variables, $V\text{-}V_1$), (also the set $R=A \cup B$, and, (iii) p (=2*number of variables not in R), consequent lists [1].

2.3 Description of 3-SAT-Satisfiability ($\mathcal{F}$) formulation:

The several features of the computations involved in 3-SAT-Satisfiability ($\mathcal{F}$) are

(i) In the Algorithm 3-SAT-Satisfiability ($\mathcal{F}$), we note that, in sub-step 1.4 of step IV, we note that $F_{x,y}$ = **false** is used in our computations to conclude that $x \wedge y$ = **false** $\Leftrightarrow \overline{x \wedge y}$ = **true**; hence we have discovered the two-clause ($\bar{x} \vee \bar{y}$).

(ii) The use of $W_{x,y,}$ coupled with the propagation of discovered restrictions on literals at upper levels, as done in Algorithm 3-SAT-Satisfiability($\mathcal{F}$), guarantees the completeness of the discovery of restrictions on literals due to clause satisfactions.

(iii) We note that, our computations proceed by considering the conditional, $a \rightarrow b$, as well as its contra positive $\bar{b} \rightarrow \bar{a}$. For the purpose of construction of the literal sets, its significance is as follows. When $W_{a,a}$ includes a literal b, we have $W_{a,a} \supseteq W_{b,b}$; however, for the "negated" (dual) literals, we also simultaneously require, in our analysis, $W_{\bar{b},\bar{b}} \supseteq W_{\bar{a},\bar{a}}$. Thus, the contra positive closure (step 3) operation translates to the following.

When there are no changes in this process of expanding the literal sets, our analysis terminates. At this termination, if *Satisfiable* = **true**, then satisfying truth-value assignment exists [1].

Number of operations needed by W-Closure (A) have bound of O (n^5) and the algorithm 3-SAT-Satisfiability (F) has a bound of O (n^{13}) {=O (n^5) x O (n^6) x O (n^2)} [1].

We have modified 3-SAT-Satisfiability (F) algorithm [1] to include step 0. Step 0 discusses mainly two properties.

- Row analysis to discover the stable literals:
  (Reduction) $(\neg a \vee \neg b \vee d) \wedge (\neg a \vee b \vee d) \equiv (\neg a \vee d)$
- Symmetry check and symmetry establishment, if needed
  The matrix W is symmetric, i.e., $W_{a,b} = W_{b,a}$. (follows from the commutativity of $\wedge$)

Chapter 3
Implementation and Testing of Sequential Algorithm

We implemented the sequential algorithm in C++ and tested on datasets [5]. The number of variables and required time is listed in the below table.

No. of Variables	No. of Clauses	Time (in seconds)
3	1	0
3	4	0
3	8	0
10	43	1
20	86	5
40	129	36
40	172	27
50	215	1046
60	258	348
70	301	155
80	344	801
90	387	1869
100	430	17621

Table 1: Number of variables and required time in sequential implementation

This is the time required for different datasets when executed on a machine of configuration as shown below

Manufacture: Dell

Model: Inspiron1464

Processor: Intel(R) Core(TM)i3 CPU M 330 @2.13GHz 2.13 GHz

Installed memory (RAM): 4.00GB (3.80 GB usable)

The following graph describes about No. of variables vs. Time (in seconds) based on the Table 1

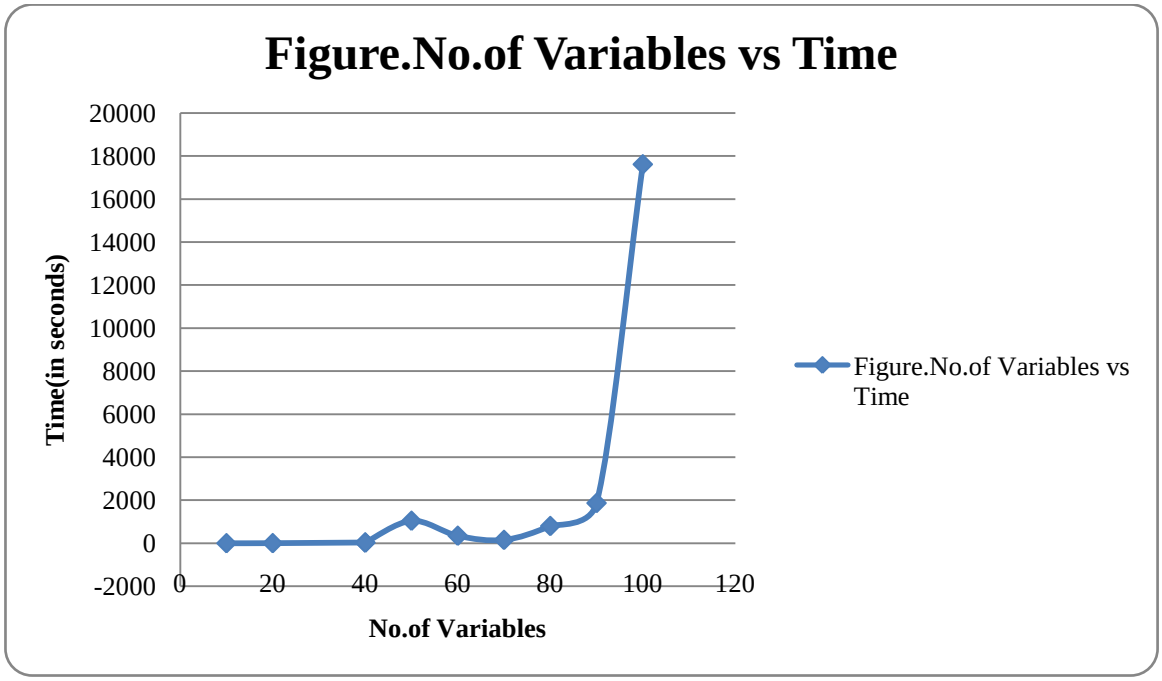

Fig 2: No. of Variables vs. Time in Sequential implementation

The graph in Fig 2 shows that after 90 variables the required time increases rapidly because it has more number of clauses and the nature of clauses (the type of clauses leads to restriction of truth values). The performance of 3-SAT problem depends upon both number of variables and clauses. This algorithm performance is also based on the type of clauses because of this claim, the 3-SAT problem requires us to satisfy a clause, say *C*, in the context of all the other (satisfied) clauses. This constraint may result in the restriction of the truth-value assignment(s). This restriction is formulated as $C \rightarrow s$.

By observing the Fig 2, required time is more for large datasets. So, to speed up the computations, we identified inherit parallelism in the portion of the algorithm. Subsequently the algorithm is converted to parallel algorithm that is reported in chapter 4.

The results for some of the datasets are included as appendices in this report. In the results, we are also able get truth value assignments for all the variables of a 3-SAT formula and those are included in appendices.

Chapter 4

PRAM Algorithm for 3-SAT

4.1 PRAM model of parallel computation:

Parallel Random Access Machine (PRAM) is a popular model for writing parallel algorithms. It consists of a number of *processors* that have a common *shared memory*.

- Shared memory model with unbounded set of processors (RAM) having own memory and usual operations and instructions.
- The cost of arithmetic operation is constant.
- Each processor is indexed by a natural number.
- Computation starts with single active processing element with input stored in global memory, log p time needed to activate p processors.
- Any number of processors can read from the same memory location simultaneously.

The operation of a synchronous PRAM can result in simultaneous access by multiple processors to the same location in shared memory. There are several variants of PRAM model, depending on whether such simultaneous access is permitted (concurrent access) or prohibited (exclusive access). As accesses can be reads or writes, we have the following four possibilities [10]:

1. Exclusive Read Exclusive Write (EREW): This PRAM variant does not allow any kind of simultaneous access to a single memory location. All correct programs for such a PRAM must insure that no two processors access a common memory location in the same time unit.

2. Concurrent Read Exclusive Write (CREW): This PRAM variant allows concurrent reads but not concurrent writes to shared memory locations. All processors concurrently reading a common memory location obtain the same value.

3. Exclusive Read Concurrent Write (ERCW): This PRAM variant allows concurrent writes but not concurrent reads to shared memory locations. This variant is generally not considered independently, but is subsumed within the next variant.

4. Concurrent Read Concurrent Write (CRCW): This PRAM variant allows both concurrent reads and concurrent writes to shared memory locations. There are several sub-variants within this variant, depending on how concurrent writes are resolved.

(a) Common CRCW: This model allows concurrent writes if and only if all the processors are attempting to write the same value (which becomes the value stored).

(b) Arbitrary CRCW: In this model, a value arbitrarily chosen from the values written to the common memory location is stored.

(c) Priority CRCW: In this model, the value written by the processor with the minimum processor id writing to the common memory location is stored.

(d) Combining CRCW: In this model, the value stored is a combination (usually by an associative and commutative operator such as or max) of the values written.

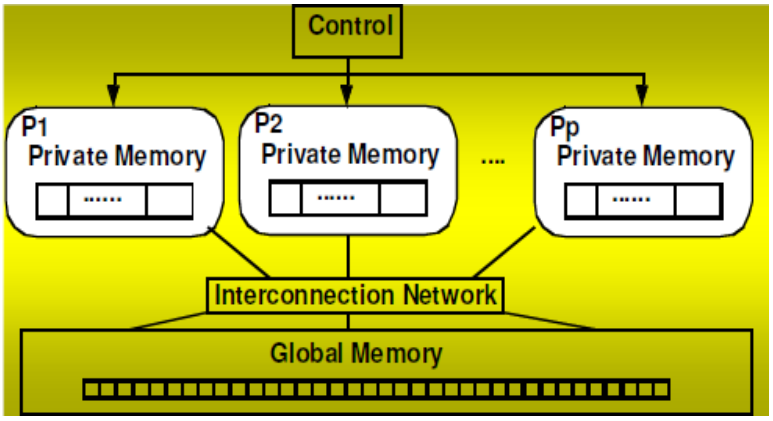

Fig 3: PRAM model of parallel computation

The two popular statements used in PRAM model of parallel computation to describe algorithm are

1. spawn (< processor names>) (log p) time needed
2. for all <processor list> do <{statement name} end for

These two statements describe, the given work is spawned among available processors and each processor does assigned work.

4.2 PRAM Algorithm for 3-SAT:

This PRAM Algorithm for our 3-SAT problem is based on the Concurrent Read Concurrent Write (CRCW) model as we always tries to set the bit in all our fields but we never tries to reset it if it is already set.

PRAM Algorithm: *W*-Closure (*A*);
1. Initialize $B = \Phi$; Satisfy-Flag = **true**;
2. Repeat steps 2.1 to 2.5 below (until there is no update in *B*) {

2.1 b-update = false; $R = A \cup B$; // working set of restricted literals

2.2 Repeat steps 2.2.1 to 2.2.3 below (until there is no change in R) {

2.2.1 if there exists a literal as well as its negation in R, then

{Satisfy-Flag = **false**; exit W-Closure; }

2.2.2 else (*i.e.*, if there does not exist a literal as well as its negation in R)

Spawn $(P_1, P_2, \ldots\ldots P_x)$

For all P_i, ***where*** $1 \leq i \leq x$, $(x \leq 2n)$ ***do***

{ $B = B \cup W_{m,l}$, for $m, l \in R$; // .. update B, R to include the literals n such that $R = R \cup B$; } // .. $n \in W_{m,l}$, for $m, l \in R$

2.2.3 } until there is no change in R;

2.3 For all literals c (c in { $c_1, c_2, \ldots, c_p$ }, p = 2*number of variables not in R) we initialize the p lists, $L_{c1}, L_{c2}, \ldots, L_{cp}$, as follows: //initialization is done in parallel.

Spawn $(P_{c1}, P_{c2}, \ldots\ldots P_{cp})$

For all P_{ci}, ***where*** $1 \leq i \leq p$ ***do***

$L_{ci} = \{n,$ a literal of variable not in $R \mid n \in W_{x,ci}$, for some $x \in R\}$ $(i = 1, \ldots, p)$;

2.4 Repeat each step 2.4.1 to 2.4.4 in parallel below (until there is no change in the lists L_{ci} $(i = 1, \ldots, p)$, or b-update)

Spawn $(P_{c1}, P_{c2}, \ldots\ldots P_{cp})$

For all P_{ci}, ***where*** $1 \leq i \leq p$ ***do***

2.4.1 For all l in L_{ci}, include $\bar{c}i$ in $L_{\bar{l}}$; // .. include contrapositives

Spawn $(P_{c1}, P_{c2}, \ldots\ldots P_{cp})$

For all P_{ci}, ***where*** $1 \leq i \leq p$ ***do***

2.4.2 Update list L_{ci} to include the literals n (of variables not in R), such that $n \in W_{m,l}$, for some $m, l \in R \cup L_{ci} \cup$ ci;

Spawn $(P_{c1}, P_{c2}, \ldots\ldots P_{cp})$

For all P_{ci}, ***where*** $1 \leq i \leq p$ ***do***

2.4.3 if there exists k, such that L_k contains a literal as well as its negation, then { update B,R to include k^-; b-update = **true** };

2.4.4 } until there is no change in the lists L_{ci} $(i = 1, \ldots, p)$, $\vee$ (b-update);

2.5 } Until ((not (Satisfy-Flag $\wedge$ b-update));

3. Return *Satisfy-Flag*, B, R, and p (= 2*number of variables not in R) lists L_{ci} $(1 \leq i \leq p)$.

PRAM Algorithm: 3-SAT-Satisfiability (F);

I. For all literal-pairs (a, b) in $\mathcal{L} \times \mathcal{L}$, let $C_{a,b}$ denote the list of clause consequents of the antecedent literal pair (a, b). Initialize $W_{a,b}$ to $C_{a,b} \cup \{a, b\}$. In this initialization, we note that $W_{a,a} = \{a\}$. For all literal pairs (a, b), initialize $F_{a,b}$ to truth-value **true**.

II. Declare (and initialize) global variables needed for *W*-Closure( ). Thus, declare and initialize *B*, set of literals, to null-set Φ. Initialize *Satisfy-Flag* to **true**. For all literals *a* in $\mathcal{L}$, initialize lists L_a to null-set Φ.

III. Let *T* be list of (top level) restricted literals; initialize *T* to null-set Φ. Initialize *Satisfiable* = **true**.

IV. Repeat steps 0,. . ,5 (until there is no change, update in *T*) { *Change* = **false**;

0. ***Spawn*** $(P_1,P_2,\ldots\ldots P_{2n})$
For all P_x, ***where*** $x \in$ ***all literals in L ,do***
0.a. For all literals *x* in *L* { // ... row analysis to discover stable literals
0.a.1. For all literals *y* in $L - \{x\}$ {
0.a.5. if $(W_{x,y} \cap W_{x,\bar{y}}) \not\subset W_{x,x}$ then {
a.5.1. $W_{x,x} = W_{x,x} \cup (W_{x,y} \cap W_{x,\bar{y}})$;
a.5.2. *Change* = **true**;
a.5.3. }; … end of if in step 0.a.2
0.a.6. }; // … end of For loop in step 0.a.1
0.a.7. }; // … end of For loop in step 0.a

Spawn $(P_1,P_2,\ldots\ldots P_{4n})$
For all P_x, ***where*** $x \in$ ***all literals in L ,do***
0.b. For all literal pairs (x, y) in $\mathcal{L} \times \mathcal{L}$ with $x < y$, {
// ... symmetry check and symmetry establishment, if needed
0.b.1. if $W_{x,y} \neq W_{y,x}$ then {
0.b.2. $W_{x,y} = W_{x,y} \cup W_{y,x}$;
0.b.3. $W_{y,xy} = W_{x,y}$;
0.b.4. *Change* = **true**;
0.b.5. }; // … end of if in 0.c.1
0.b.6. }; // . end of For loop in step 0.c

1. Spawn $(P_1,P_2,\ldots\ldots P_{4n})$
For all P_x, ***where*** $x \in 1 \leq i \leq 4n$
For all literal pairs (x, y) in $\mathcal{L} \times \mathcal{L}$, whose variables not in *T* , { // ... literal-pair analysis
1.1. initialize (the lists, or sets) $W_{x,y} = T \cup W_{x,x} \cup W_{y,y} \cup W_{x,y}$;
1.2. invoke *W*-Closure($W_{x,y}$); // .. results in *Satisfy-Flag*,
// .. and *B*, set of restricted literals
1.3. if (*Satisfy-Flag*) then $W_{x,y} = W_{x,y} \cup B$ // .. include restricted literals
1.4. else {
1.4.1. $F_{x,y}$ = **false**; // .. $x \wedge y$ = **false** $\Leftrightarrow \overline{x \wedge y}$ = **true**;
1.4.2. if $\bar{y} \notin W_{x,x}$ then { $W_{x,x} = W_{x,x} \cup \{\bar{y}\}$; *Change* = **true**; }
1.4.3. if $\bar{x} \notin W_{y,y}$ then { $W_{y,y} = W_{y,y} \cup \{\bar{x}\}$; *Change* = **true**; }
1.4.4. }; // .. end of else in step 1.4
1.5. }; // .. end of For loop in step 1

2. Spawn $(P_1,P_2,\ldots\ldots P_{2n})$
For all P_x, ***where*** $x \in 1 \leq i \leq 2n$ ***do {*** // .. Top level restricted literals
2.1. if $\bar{x} \notin T$ and $W_{x,x}$ includes a literal as well as its negation {

2.2. $F_{x,x}$ = **false**;
2.3. $T = T \cup \{ \bar{x} \}$;
2.4. invoke *W*-Closure(*T*); // .. results in *Satisfy-Flag*, and *B*, set of literals
2.5. $T = T \cup B$; // .. include restricted literals
2.6. if ((not *Satisfy-Flag*)) then {
2.6.1. *Satisfiable* = **false**;
2.6.2. Exit For loop in step 2 & loop in step IV
2.6.3. }; // .. end of if of step 2.6
2.7. *Change* = **true**; }; // .. end of if in step 2.1
2.8. } ; // .. end of For loop in step 2

3. Spawn ($P_1, P_2, \ldots\ldots P_{2n}$)
For all P_x, where $x \in 1 \le i \le 2n$ do { // .. Contrapositive closure
3.1. if (($\bar{x} \notin W_{\bar{a},\bar{a}}$) and ($a \in W_{x,x}$)) then {
3.2. $W_{\bar{a},\bar{a}} = W_{\bar{a},\bar{a}} \cup \{ \bar{x} \}$;
3.3. invoke *W*-Closure($W_{\bar{a},\bar{a}}$); // .. results in *Satisfy-Flag*, and *B* literals
3.4. if (*Satisfy-Flag*) then $W_{\bar{a},\bar{a}} = W_{\bar{a},\bar{a}} \cup B$; // .. include restricted literals
3.5. if ((not *Satisfy-Flag*)) then {
3.5.1. $F_{\bar{a},\bar{a}}$ = **false**;
3.5.2. $T = T \cup \{a\}$;
3.5.3. invoke *W*-Closure(*T*); // .. results in *Satisfy-Flag*, and *B*
3.5.4. $T = T \cup B$;
3.5.5. if ((not *Satisfy-Flag*)) then {
3.5.5.1. *Satisfiable* = **false**;
3.5.5.2. Exit For loop in step 3 & loop in step IV
3.5.5.3. }; // .. end of if in step 3.5.5
3.6. }; // .. end of if of step 3.5
3.7. *Change* = **true**; }; // .. end of if in step 3.1
3.8. } ; // .. end of For loop in step 3

4. Spawn($P_1, P_2, \ldots\ldots P_{2n}$)
For all P_x, where $x \in 1 \le i \le 2n$ do { // .. Transitive closure
4.1 For all literals $a \in W_{x,x}$ such that $W_{a,a} \not\subset W_{x,x}$ {
4.2 $W_{x,x} = W_{x,x} \cup W_{a,a}$; // .. transitivity closure for the case $W_{a,a} \not\subset W_{x,x}$
4.3 invoke *W*-Closure($W_{x,x}$); // .. results in *Satisfy-Flag*, and *B*, set of literals
4.4 if (*Satisfy-Flag*) then $W_{x,x} = W_{x,x} \cup B$; // .. include restricted literals
4.5 if ((not *Satisfy-Flag*)) then {
4.5.1 $F_{x,x}$ = **false**;
4.5.2 if $\bar{x} \notin T$ then {
4.5.3 $T = T \cup \{ \bar{x} \}$;
4.5.4 invoke *W*-Closure(*T*); // .. results in *Satisfy-Flag*, and *B*
4.5.5 $T = T \cup B$;
4.5.6 if ((not *Satisfy-Flag*)) then {
4.5.6.1 *Satisfiable* = **false**;
4.5.6.2 Exit For loop in step 4 & loop in step IV
4.5.6.3 }; // .. end of if in step 4.5.6
4.5.7 }; // .. end of if of 4.5

4.6 .*Change* = **true**; }; // end of For loop in step 4.1

4.7 } ; // .. end of For loop in step 4

5. Until (not (*Satisfiable* $\wedge$ *Change*)); // .. end of IV

V. If *Satisfiable* then *T* contains list of top-level restricted literals, $W_{x,x}$ contains set of restricted literals when x = **true** and $W_{x,y}$ contains set of restricted literals when the pair of literals (x , y) = (**true, true)**.

4.3 PRAM Time Complexity:

a) PRAM Algorithm: *W*-Closure (*A*):-

Our analysis for 3-SAT-satisfiability progresses by updating $W_{a,b}$ as reported in [2]. Hence, we require *W*-Closure (*A*) operation. Some steps in W-Closure (A) can be parallelized, all steps in sub steps in 2.2 are parallelized, step 3 is parallelized and all the sub steps in 2.4 are parallelized.

In Step 2.4, the steps 2.4.1 to 2.4.3 are repeated till there is no change in the lists and each step from 2.4.1 to 2.4.3 are parallelized. During this repetition, we note that literal(s) are only added (and never deleted), and each list can grow to include maximum p = 2*number of variables not in *R*) number of literals. Since we focus on demonstration of time complexity, we only consider an upper-bound on the length of list as O (n). In step 2.4, we maintain p = 2*number of variables not in *R*) number of lists, again we may treat this expression as O (n) lists.

In step 2.4.2, for each list, we need to consider pairs of literals; since the list size is O (n), the number of ordered pairs is O (n^2). For each ordered pair, we need to retrieve $C_{m,l}$ (or $W_{m,l}$, in case of *W*-Closure), whose size has an upper bound of O(n). This makes a count of O (n^2) time complexity with O (n^2) number of processors (each processor will take one order pair). In step 2.4, we maintain O (n) lists, and hence the count of the number of operations is O (n^3) with O (n^2) number of processors.

We note that, from amongst the steps 2.1 to 2.5 that are repeated for completing step 2, the costliest step is 2.4, and it requires O (n^3) operations with O (n^2) number of processors.

Steps 2.1 to 2.5 are repeated for each possible change in *B*; there are O (n) changes possible in *B*. Hence the total number of operations required for completing step 2 is O (n^4) with O (n^2) number of processors.

Since Step 2 requires maximum number of operations in the algorithm PRAM W-Closure (*A*), we conclude that the number of operations needed by PRAM W-Closure (*A*)) have a bound of O (n^4) time complexity with O (n^2) number of processors.

b) PRAM Algorithm: 3-SAT-Satisfiability (F):-

Some steps in 3-SAT Satisfiability algorithm can be parallelized. Steps 0,1,2,3 and 4 are parallelized.

W-Closure (*A*) is invoked in Step IV's sub step 1.2 of Algorithm: 3-SAT-Satisfiability ($\mathcal{F}$). The step 1 involves construction of multiple lists, one for each the literal-pair (*a*, *b*). An upper bound the number of these literal pairs is O (n^2). These literal pairs distributed among available number of processors, an upper bound the number of processors are O (n^2) in step1.

In Step IV, the steps 1 to 5 are repeated for each change. An upper bound for these changes are O (n^6) [1]. During every change, we execute sub-step 1 of step IV of Algorithm: 3-SAT-Satisfiability ($\mathcal{F}$).

In sub-step 1, for each literal pair (*a*, *b*), we invoke W-Closure ($W_{a,b}$). We noted that each invocation of *W*-Closure ($W_{a,b}$) has a bound of O (n^4) operations with O (n^2) number of processors. Hence, step 1 has a bound of O (n^4) operations with O (n^3) number of processors (each processor will take one literal pair).

As we noted, the bound on the number of repetitions of sub step 1 is O (n^6). This combined with the requirement of O (n^4) operations per repetition gives us a bound of O (n^{10}) operations with O (n^3) number of processors. Hence the bound on the total number of operations needed by step IV is O (n^{10}) with O (n^3) number of processors. Since Step IV is the step requiring maximum number of operations for the Algorithm: 3-SAT-Satisfiability ($\mathcal{F}$), this serves as a bound on our Algorithm: 3-SAT-Satisfiability ($\mathcal{F}$) as well.

Chapter 5

Experimental Test bed

5.1 Software

We implemented the sequential algorithm by using *GCC* compiler and implemented the PRAM algorithm by using OpenMP compiler directives available in *GCC* and it can also be implemented by using MPI.

5.1.1 OpenMP

An Application Program Interface (API) that may be used to explicitly direct multi-threaded, shared memory parallelism. OpenMP provides a portable, scalable model for developers of shared memory parallel applications. The API supports C/C++ and FORTRAN on multiple architectures, including UNIX & Windows NT [9].

OpenMP Comprised of three primary API components:

- Compiler Directives
- Runtime Library Routines
- Environment Variables

a) C / C++ Directives Format:-

In C/C++, we have to include omp.h header file to write Open MP directives.

Format: #pragma omp [directive name] [clauses,….]

Example: #pragma omp parallel default (shared) private (beta, pi)

b) PARALLEL directive:

A parallel region is a block of code that will be executed by multiple threads.

- When a thread reaches a PARALLEL directive, it creates a team of threads and becomes the master of the team. The master is a member of that team and has thread number 0 within that team.
- Starting from the beginning of this parallel region, the code is duplicated and all threads will execute that code.
- There is an implied barrier at the end of a parallel section. Only the master thread continues execution past this point.
- If any thread terminates within a parallel region, all threads in the team will terminate, and the work done up until that point is undefined.

c) DO / for Directive

The DO / for directive specifies that the iterations of the loop immediately following it must be executed in parallel by the team. This assumes a parallel region has already been initiated, otherwise it executes in serial on a single processor.

#pragma omp for *[clause ...]*

d) Combined Parallel Work-Sharing Constructs

- PARALLEL DO / parallel for
- PARALLEL SECTIONS

For the most part, these directives behave identically to an individual PARALLEL directive being immediately followed by a separate work-sharing directive. Most of the rules, clauses and restrictions that apply to both directives are in effect.

e) PRIVATE Clause

The PRIVATE clause declares variables in its list to be private to each thread. PRIVATE variables behave as follows:

- A new object of the same type is declared once for each thread in the team.
- All references to the original object are replaced with references to the new object.
- Variables declared PRIVATE should be assumed to be uninitialized for each thread.

f) SHARED Clause

The SHARED clause declares variables in its list to be shared among all threads in the team.

- A shared variable exists in only one memory location and all threads can read or write to that address.
- It is the programmer's responsibility to ensure that multiple threads properly access SHARED variables (such as via CRITICAL sections).

g) DEFAULT Clause

The DEFAULT clause allows the user to specify a default scope for all variables in the lexical extent of any parallel region.

- Specific variables can be exempted from the default using the PRIVATE, SHARED, FIRSTPRIVATE, LASTPRIVATE, and REDUCTION clauses
- The C/C++ OpenMP specification does not include private or firstprivate as a possible default. However, actual implementations may provide this option.

- Using NONE as a default requires that the programmer explicitly scope all variables.

5.1.2 MPI (Message Passing Interface)

- A **message-passing library** specification
  - Not a compiler specification
  - Not a specific product
- Used for parallel computers, clusters, and heterogeneous networks as a message passing library
- Designed to be used for the development of parallel software libraries, i.e. Star-MPI
- Designed to provide access to advanced parallel hardware for
  - End users
  - Library writers
  - Tool developers

Two primary mechanisms needed:

1. A method of creating separate processes for execution on different computers
2. A method of sending and receiving messages

Some Basic Concepts [12]

- Processes can be collected into groups
- Each message is sent in a context, and must be received in the same context
- A group and context together form a communicator
- A process is identified by its rank in the group associated with a communicator
- There is a default communicator whose group contains all initial processes, called *MPI_COMM_WORLD*

Begin programming with 6 MPI function calls:

- MPI_INIT *Initializes MPI*
- MPI_COMM_SIZE *Determines number of processes*
- MPI_COMM_RANK D*etermines the label of the calling process*
- MPI_SEND *Sends a message*
- MPI_RECV *Receives a message*
- MPI_FINALIZE *Terminates MPI*

MPI Basic Send/Receive

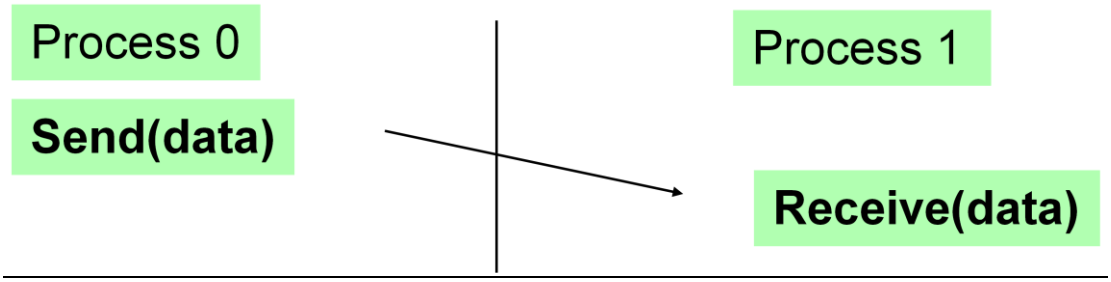

Fig 4: MPI Basic Send/Receive

int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest,
int tag, MPI_Comm comm)

Source process sends message to *destination* process

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source,
int tag, MPI_Comm comm, MPI_Status *status)

Destination process receives a message from *source*

Input Parameters

buf	initial address of send buffer (choice)
count	number of elements in send buffer (nonnegative integer)
datatype	datatype of each send buffer element (handle)
dest	rank of destination (integer)
tag	message tag (integer)
comm	communicator (handle)

5.1.3 GCC (GNU Compiler Collection)

The **GNU Compiler Collection** (**GCC**) is a compiler system produced by the GNU Project supporting various programming languages like C, C++, FORTRAN, Pascal, Objective-C, Java, and Ada, among others. GCC is free software, distributed under the GNU General Public License (GNU GPL). This means you have the freedom to use and to modify GCC, as with all GNU software. If you need support for a new type of CPU, a new language, or a new feature you can add it yourself, or hire someone to enhance GCC for you. You can hire someone to fix a bug if it is important for your work.

GCC supports both OpenMP programming and MPI programming. GCC library has omp.h header file and it contains OpenMP compiler directives, runtime library routines and environment variables.

The following syntax is for compilation and execution of OpenMP programs in GCC

Compilation: gcc -fopenmp <Program name> -o <executable file name>
Run: <executable file>

GCC library also has mpi.h header file and helps to write MPI programs.

The following is for compilation and execution of MPI programs in GCC

Compilation: mpicc <program name> -o <executable file name>
Run: mpirun -np <number of processors> <executable file>

5.2 Hard ware

Multi-core processor:

Processors were originally developed with only one **core**. The core is the part of the processor that actually performs the reading and executing of instructions. A **multi-core processor** is a single component with two or more independent actual processors (called "cores"). Manufacturers typically integrate the cores onto a single integrated circuit die (known as a chip multiprocessor or CMP), or onto multiple dies in a single chip package.

The improvement in the performance gained by the use of a multi-core processor depends very much on the software algorithms used and their implementation. In particular, possible gains are limited by the fraction of the software that can be parallelized to run on multiple cores simultaneously; this effect is described by Amdahl's law.

We have used a machine of configuration as shown below

Manufacture: Dell

Model: Inspiron1464

Processor: Intel(R) Core(TM) i3 CPU M 330 @2.13GHz 2.13 GHz

Installed memory (RAM): 4.00GB

And we have also used a super computing machine available at CMSD

Manufacture: IBM p595

Processor: 64-CPU with Power5 @ 1.9 GHz

Installed memory (RAM): 128 GB RAM

Chapter 6

Parallel Implementation and Testing of PRAM Algorithm for 3-SAT

We implemented the parallel algorithm using Open MP directives; Open MP is a shared memory programming. We tested on datasets [5]. The number of variables and required time is listed in the below table.

No. of Variables	No. of Clauses	Time (in seconds)
3	1	0
3	4	0
3	8	0
10	43	0
20	86	1
40	129	4
40	172	20
50	215	551
60	258	174
70	301	143
80	344	641
90	387	1434
100	430	6251

Table 2: Number of variables and required time in parallel implementation

This is the time required for different datasets when executed on a machine of configuration as shown below

Manufacture: Dell
Model: Inspiron1464
Processor: Intel(R) Core(TM) i3 CPU M 330 @2.13GHz 2.13 GHz
Installed memory (RAM): 4.00GB (3.80 GB usable)

The following graph describes about No. of variables vs. Time (in seconds) based on the Table 2

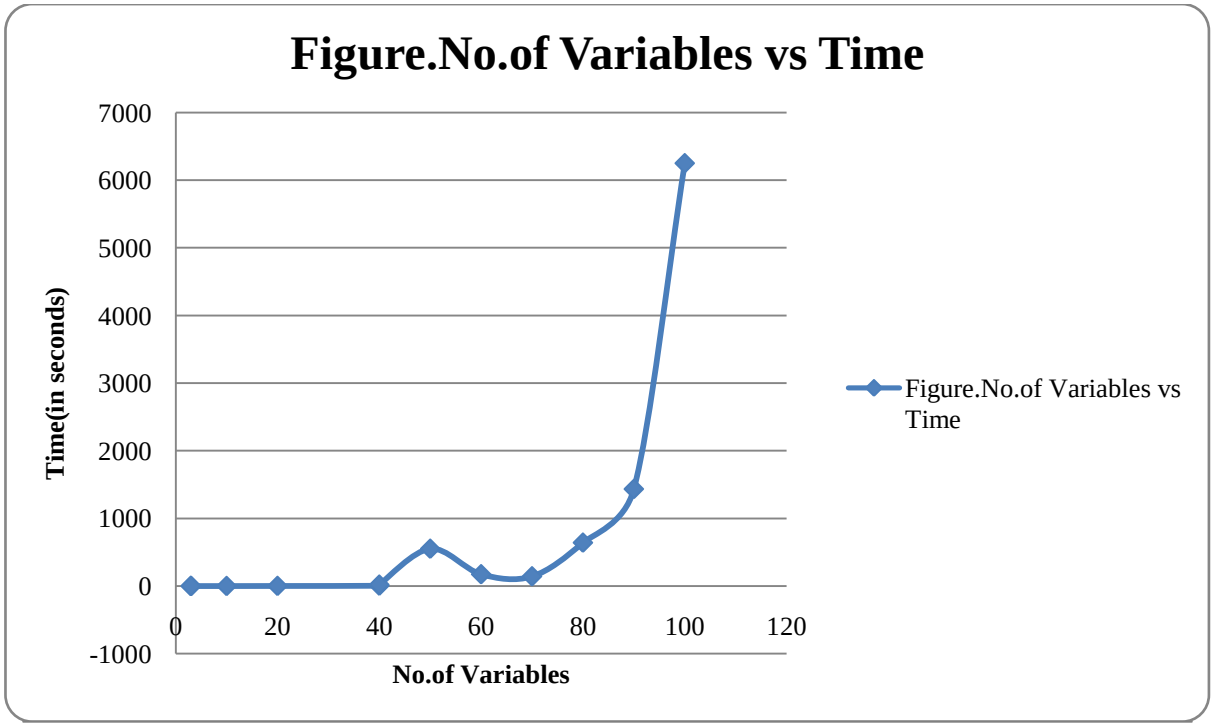

Fig 5: No. of Variables vs. Time in parallel implementation

The graph in Fig 2 shows that after 90 variables the required time increases rapidly because it has more number of clauses and the nature of clauses (the type of clauses leads to restriction of truth values) as we discussed in chapter 3.

The timing requirement for testing the 100 variables in sequential implementation is 293 minutes (fig 2) where as the timing requirement for testing the same dataset in parallel implementation is 104 minutes (fig5).

Chapter 7

Conclusion and Future Work

7.1. Conclusion

We have implemented both sequential and parallel implementation of 3-SAT algorithm [1], tested on datasets [5] and results are reported. We have also implemented the Truth-Value assignment algorithm for 3-SAT problem which gives complete truth value assignment for all the variables of the given formula if formula is satisfiable.

Amdahl's law is used in parallel computing to predict the maximum speed up using multiple processors. Speed up (S) = $\frac{1}{(1-P)+\frac{P}{N}}$

Where P is the portion of parallelism, N is the number of processors.

The speed up of our problem using multiple processors is shown in Fig 6.

7.2.Future Work

Implementation of PRAM algorithm for 3-SAT problem can be carried out by using distributed memory programming kind of paradigm like MPI.

Instead of Matrix representation for 3-SAT Algorithm [1], pair representation is possible with an upper bound of O (n^3 * k^3) where n is the number of variables and k is number of clauses. This representation results in reduction of both space complexity and time complexity.

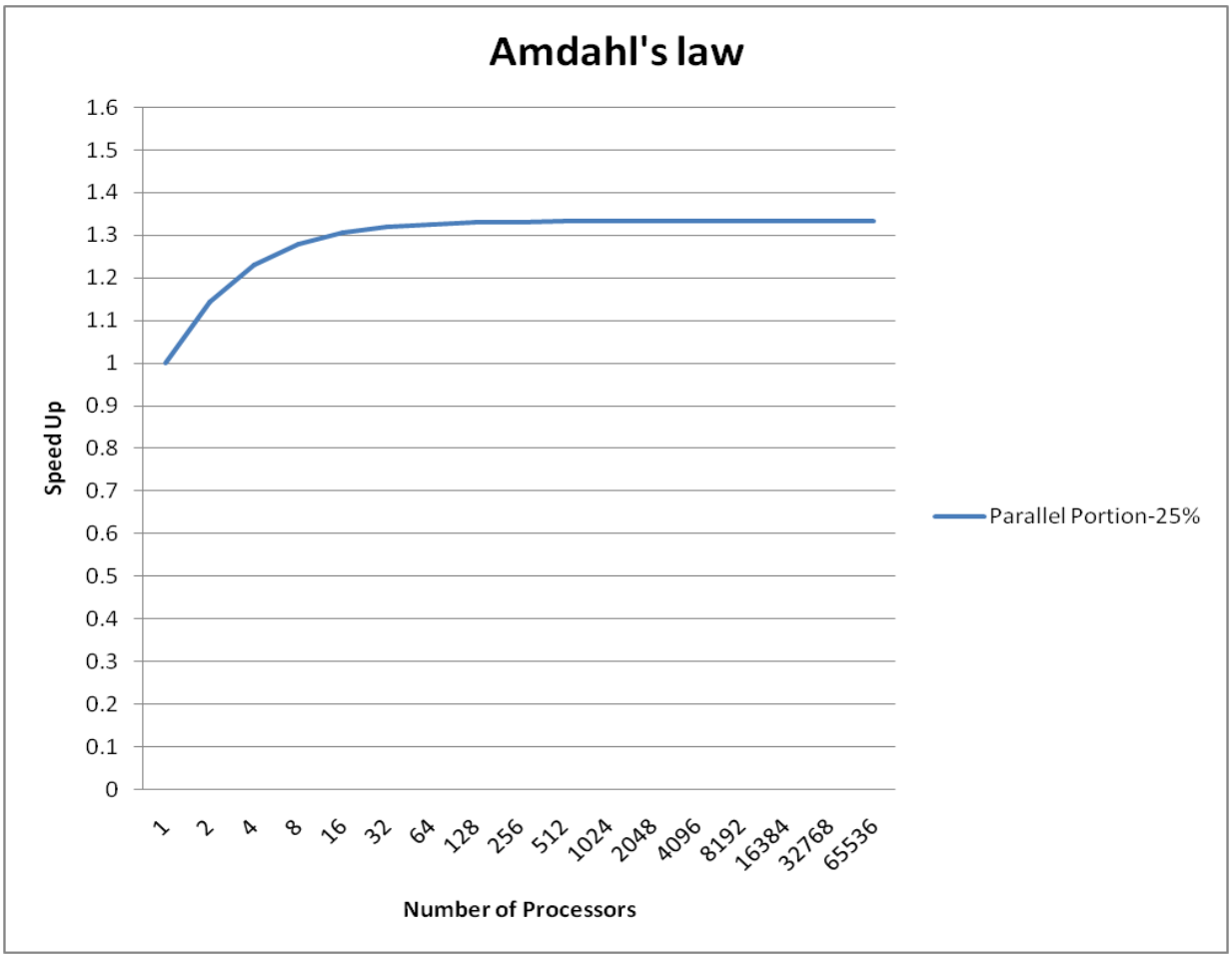

Fig 6: Speed Up vs. Number of Processors using Amdahl's law with P=25%

Fig 6 shows the graph between Speed Up and Number of processors using Amdahl's law with 25% of parallelism. The Speed Up is increasing gradually up to some number of processors and then it is constant because of embarrassingly parallel problem [11].

References

[1] N.S. Chaudhari, Computationally Hard Problems*:* 3-SAT and its polynomial solvability Jour. Ind. Acad. Maths, 31(2) (2009).

[2] N.S. Chaudhari, "Improved polynomial algorithm for 3-SAT", Jour. Ind. Acad. Maths, Vol. 32, No.1 (Oct. 2010) pp. 251-267.

[3] J.E. Hopcroft, R. Motwani, and J.D. Ullman, Automata, Languages, and Computation, third ed., pp. 458, (Exercise 10.3.4), Addison Wesley, N.Y., 2007.

[4] Aho, A.V., Hopcroft, J.E., and Ullman, J.D.: (1974) The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA.

[5] http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html.

[6] Wikipedia: http://en.wikipedia.org/wiki/DPLL_algorithm (Retrieved 09 May 2009: version updated as on 01 April 2009).

[7] Wikipedia (downloaded on 22nd Aug., 2010)
http://en.wikipedia.org/wiki/P_versus NP_problem.

[8] Deolalikar Vnay. (2010, August). posted on the websites
http://www.hpl.hp.com/personal/Vinay_ Deolalikar/ ,
http://www.scribd.com/doc/35539144/pnp12pt,
http://www.hpl.hp.com/news/.

[9] Using OpenMP, Portable Shared Memory Parallel Programming, Barbara Chapman, Gabriele Jost and Ruud van der Pas.

[10] Efficient parallel algorithms By Alan Gibbons, Wojciech Rytter.

[11] Amdahl's law, http://en.wikipedia.org/wiki/Amdahl%27s_law.

[12] Using MPI, 2nd Edition, Portable Parallel Programming with the Message Passing Interface, William Gropp, Ewing Lusk and Anthony Skjellum.

1. Datasets

Dataset-1: 3-variables 4-clauses

1) 1 2 3
2) -1 2 3
3) 1 -2 3
4) -1 -2 3

Dataset-2: 3-variables 8-clauses

1) -1 -2 -3
2) -1 -2 3
3) -1 2 -3
4) -1 2 3
5) 1 -2 -3
6) 1 -2 3
7) 1 2 -3
8) 1 2 3

Dataset-3: 10-variables 43-clauses

1) -7 6 -9
2) 7 -5 9
3) 3 7 8
4) -5 2 7
5) 9 -10 2
6) -6 7 3
7) 2 -10 1
8) -1 8 4
9) 7 -5 3
10) -5 -6 -9
11) -2 4 6
12) -6 -1 -3
13) -10 5 3
14) -9 -7 5
15) 10 -9 2
16) -1 10 -9
17) 10 6 7
18) -7 9 -10
19) -4 -10 -6
20) 2 7 -4
21) 9 1 -4
22) 10 8 -7
23) -7 8 6
24) -5 10 -6
25) 3 5 -10
26) 6 -2 -10
27) 8 1 -3
28) -5 -2 8
29) 7 -4 -2
30) -3 -10 7
31) 9 -4 3
32) -7 -2 10
33) -4 5 6
34) -9 4 3
35) 10 -7 4
36) 9 4 8
37) -3 -6 4
38) 7 -10 -2
39) -7 -5 3
40) -2 5 -10
41) -9 4 -10
42) 5 -7 -6
43) -5 3 7

Dataset-4: 20-variables 86-clauses

1) -13 12 -19
2) 13 -10 12
3) 5 15 18
4) -10 4 16
5) 17 -19 3
6) -12 14 7
7) 4 -5 1
8) -1 17 9
9) 13 -10 6
10) -10 -13 -14
11) -3 9 14
12) -12 -1 -5
13) -19 9 7
14) -18 -13 11
15) 20 -18 3
16) -1 2 -20
17) 19 11 16
18) -14 18 -16
19) -7 -8 -13
20) 4 13 -9
21) 17 2 -8
22) 19 17 -14
23) -13 17 12
24) -9 11 -12
25) 5 11 -7
26) 12 -4 -20
27) 16 2 -7
28) -10 -3 18
29) 14 -8 -3
30) -6-20 16
31) 18 -9 5
32) -13 -5 14
33) -7 10 12
34) -17 9 5
35) 20 -14 8
36) 17 8 18
37) -5 -12 9
38) 13 -14 -5
39) -14 -10 7
40) -4 9 -5
41) -18 8 -19
42) 10 -15 -13
43) -9 6 14
44) 18 15 -5
45) -10 -1 -15
46) -3 1 -17
47) -2 14 7
48) 5 -6 12
49) 12 -16 -2
50) 4 -11 -8
51) 13 3 10
52) 20 2 -12
53) -16 7 8
54) 6 -11 17
55) -10 3 4
56) 9 18 -7
57) -2 -3 20
58) -18 15 -7
59) -9 -10-17
60) -13 10 -11
61) 10 -19 -1
62) -18 -5 -17
63) -19 -2 7
64) -4 -13 -11
65) 5 -1 -12
66) -18 12 -4
67) -19 1 3
68) -19 5 4
69) -18 10 -5
70) 8 -15 7
71) -7 11 2
72) 15 -20 9
73) -19 6 -4
74) 1 -9 19
75) 19 11 14
76) -9 19 1
77) -14 -1 -8
78) 9 10 3
79) 11 -19 2
80) -11 9 6
81) 3 12 18
82) 4 -12 15
83) -3 5 1
84) 7 -9 -10
85) -20 17 -19
86) -12 17 -14

Dataset-5: 20-variables 91-clauses

1) 4 -18 19
2) 3 18 -5
3) -5 -8 -15
4) -20 7 -16
5) 10 -13 -7
6) -12 -9 17
7) 17 19 5
8) -16 9 15
9) 11 -5 -14
10) 18 -10 13
11) -3 11 12
12) -6 -17 -8
13) -18 14 1
14) -19 -15 10
15) 12 18 -19
16) -8 4 7
17) -8 -9 4
18) 7 17 -15
19) 12 -7 -14
20) -10 -11 8
21) 2 -15 -11
22) 9 6 1
23) -11 20 -17
24) 9 -15 13
25) 12 -7 -17
26) -18 -2 20
27) 20 12 4
28) 19 11 14
29) -16 18 -4
30) -1 -17 -19
31) -13 15 10
32) -12 -14 -13
33) 12 -14 -7
34) -7 16 10
35) 6 10 7
36) 20 14 -16
37) -19 17 11
38) -7 1 -20
39) -5 12 15
40) -4 -9 -13
41) 12 -11 -7
42) -5 19 -8
43) 1 16 17
44) 20 -14 -15
45) 13 -4 10
46) 14 7 10
47) -5 9 20
48) 10 1 -19

49) -16 -15 -1
50) 16 3 -11
51) -15 -10 4
52) 4 -15 -3
53) -10 -16 11
54) -8 12 -5
55) 14 -6 12
56) 1 6 11
57) -13 -5 -1
58) -7 -2 12
59) 1 -20 19
60) -2 -13 -8
61) 15 18 4
62) -11 14 9
63) -6 -15 -2
64) 5 -12 -15
65) -6 17 5
66) -13 5 -19
67) 20 -1 14
68) 9 -17 15
69) -5 19 -18
70) -12 8 -10
71) -18 14 -4
72) 15 -9 13
73) 9 -5 -1
74) 10 -19 -14
75) 20 9 4
76) -9 -2 19
77) -5 13 -17
78) 2 -10 -18
79) -18 3 11
80) 7 -9 17
81) -15 -6 -3
82) -2 3 -13
83) 12 3 -2
84) -2 -3 17
85) 20 -15 -16
86) -5 -17 -19
87) -20 -18 11
88) -9 1 -5
89) -19 9 17
90) 12 -2 17
91) 4 -16 -5

Dataset-6: 40-variables 129-clauses

1) -19 18 -30
2) 19 -15 18
3) 7 23 28
4) -14 6 24
5) 26 -29 4
6) -17 21 10
7) 6 -7 2
8) -2 26 13
9) 20 -14 9
10) -14 -19 -21
11) -4 13 21
12) -17 -1 -8
13) -28 14 11
14) -27 -20 17
15) 30 -27 4
16) -1 3 -2
17) 29 17 25
18) -20 27 -25
19) -10 -12 -21
20) 5 20 -14
21) 25 4 -12
22) 28 25 -22
23) -20 25 19
24) -13 16 -19
25) 7 16 -10
26) 18 -5 -30
27) 24 3 -10
28) -15 -4 27
29) 20 -12 -4
30) -9 -30 25
31) 27 -13 8
32) -20 -7 22
33) -11 15 19
34) -25 13 8
35) 30 -22 12
36) 25 13 28
37) -7 -18 14
38) 19 -21 -7
39) -21 -16 10
40) -6 14 -7
41) -26 13 -29
42) 15 -22 -19
43) -13 9 22
44) 27 23 -7
45) -14 -2 -23
46) -4 2 -26
47) -3 20 10
48) 7 -9 18
49) 18 -24 -3
50) 6 -16 -12
51) 20 5 15
52) 30 2 -18
53) -24 10 12
54) 9 -17 27
55) -14 4 6
56) 14 27 -11
57) -2 -5 3
58) -27 23 -10
59) -13 -15 -27
60) -19 15 -17
61) 14 -29 -1
62) -27 -8 -25
63) -28 -3 10
64) -6 -19 -17
65) 7 -1 -19
66) -27 18 -6
67) -29 1 5
68) -28 7 6
69) -26 14 -8
70) 11 -23 30
71) -11 17 2
72) 23 -22 13
73) -29 9 -5
74) 2 -13 29
75) 29 17 21
76) -13 28 1
77) -21 -1 -12
78) 14 15 4
79) 16 -29 3
80) -17 14 9
81) 5 18 28
82) 6 -18 23
83) -4 8 1
84) 10 -13 -15
85) -30 26 -29
86) -17 25 -21
87) 11 -18 -17
88) -28 14 21
89) 8 -29 -19
90) -16 28 1
91) -3 16 -5
92) -19 8 4
93) 9 -8 -5
94) 5 -24 -25
95) 9 2 10
96) 22 8 -2
97) 2 14 -28
98) 2 7 14
99) -20 18 23
100) 10 13 -26
101) -6 19 13
102) 16 -18 -9
103) -4 11 -20
104) 9 -1 -26
105) 5 -3 4

106) -26 24 -29
107) 21 -7 -1
108) -24 -1 -16
109) -29 -30 -20
110) 13 -27 30
111) 26 20 3
112) -21 20 -25
113) -12 11 24
114) 16 -30 15
115) -2 8 24
116) -6 1 14
117) -10 2 -13
118) 18 11 -2
119) -16 -29 -17
120) -9 -8 -17
121) -5 24 -21
122) 27 -7 -6
123) -30 1 25
124) 6 -17 1
125) 27 20 -16
126) 7 -25 -10
127) -29 19 3
128) 20 -6 -3
129) -15 21 -10

Dataset-7: 40-variables 172-clauses

1) -25 24 -40
2) 25 -20 24
3) 9 30 38
4) -19 8 33
5) 34 -39 6
6) -23 28 13
7) 7 -9 2
8) -2 35 18
9) 26 -19 12
10) -19 -25 -28
11) -5 17 28
12) -23 -1 -11
13) -37 19 14
14) -36 -27 23
15) 39 -37 5
16) -2 3 -40
17) 38 23 33
18) -27 36 -33
19) -14 -16 -28
20) 7 27 -19
21) 33 5 -16
22) 38 34 -29
23) -26 34 25
24) -17 22 -25
25) 9 22 -13
26) 24 -7 -25
27) 32 4 -14
28) -20 -6 37
29) 27 -17 -6
30) -12 -40 33
31) 36 -17 10
32) -26 -9 29
33) -14 20 25
34) -33 18 11
35) 40 -29 16
36) 33 17 37
37) -10 -24 19
38) 26 -28 -10
39) -28 -21 13
40) -8 18 -10
41) -35 17 -39
42) 20 -30 -26
43) -17 12 30
44) 36 31 -10
45) -19 -2 -31
46) -6 2 -35
47) -4 27 13
48) 9 -12 25
49) 24 -33 -4
50) 7 -22 -17
51) 26 6 20
52) 40 3 -24
53) -32 13 16
54) 12 -23 36
55) -19 5 8
56) 18 37 -15
57) -3 -6 4
58) -36 30 -14
59) -17 -20 -36
60) -25 20 -23
61) 19 -39 -2
62) -36 -11 -34
63) -37 -4 13
64) -8 -26 -22
65) 9 -1 -26
66) -36 24 -8
67) -38 1 7
68) -37 9 8
69) -35 19 -11
70) 15 -30 14
71) -14 23 3
72) 30 -29 18
73) -38 12 -7
74) 2 -17 18
75) 38 22 28
76) -17 38 2
77) -28 -1 -16
78) 18 20 5
79) 21 -38 4
80) -22 18 12
81) 6 24 38
82) 8 -24 31
83) -5 11 1
84) 13 -18 -20
85) -40 35 -39
86) -23 34 -29
87) 15 -25 -23
88) -37 19 28
89) 10 -39 -26
90) -21 37 2
91) -4 21 -6
92) -25 10 6
93) 12 -10 -7
94) 6 -32 -34
95) 12 2 13
96) 29 11 -3
97) 2 19 -38
98) 3 10 19
99) -27 24 31
100) 13 18 -34
101) -7 26 17
102) 21 -23 -12
103) -5 14 -28
104) 12 -2 -35
105) 7 -4 5
106) -34 32 -39
107) 28 -10 -1
108) -32 -1 -22
109) -38 -39 -27
110) 17 -35 18
111) 35 27 4
112) -28 27 -34
113) -16 14 32
114) 22 -21 39
115) -2 10 32
116) -7 1 19
117) -14 3 -18
118) 23 14 -2
119) -21 -39 -23
120) -12 -11 -22

121) -7 32 -28
122) 36 -9 -8
123) -39 1 34
124) 8 -23 1
125) 36 27 -21
126) 9 -34 -14
127) -39 26 4
128) 26 -8 -3
129) -20 28 -14
130) 39 -14 -24
131) 15 31 35
132) 25 -34 -28
133) 21 -5 -34
134) -35 -21 -33
135) -17 -31 6
136) 17 34 6
137) -6 40 5
138) -11 -15 -40
139) -35 13 28
140) -10 -22 -17
141) 21 8 1
142) 12 5 10
143) 26 31 6
144) -19 24 -35
145) 2 -34 3
146) -11 25 -17
147) 28 4 38
148) 4 -34 22
149) -33 -37 27
150) 27 16 40
151) 30 27 -26
152) 9 -13 2
153) 33 -17 38
154) -2 37 20
155) 22 -13 26
156) -10 24 -21
157) -23 -36 -1
158) -19 21 -1
159) -39 12 20
160) -12 -40 -37
161) 26 -4 28
162) -37 -9 -25
163) 24 15 10
164) 36 -11 -2
165) 4 -16 -40
166) -17 -6 -9
167) 37 3 27
168) -25 14 32
169) -5 7 9
170) -21 -23 -11
171) -24 -28 -12
172) -5 -10 -14

Dataset-8: 50-variables 215-clauses

1) 36 -38 49
2) -10 -11 38
3) 17 26 5
4) -48 10 -23
5) -43 -32 6
6) -29 2 -21
7) -9 35 44
8) 26 21 48
9) -38 26 44
10) 24 -3 -28
11) -38 -16 -31
12) 12 35 21
13) -43 -16 31
14) 29 -12 50
15) 10 -19 8
16) -28 -43 -24
17) -12 -46 28
18) 47 -19 -12
19) 30 -43 41
20) -19 35 33
21) 7 -9 -8
22) -45 -20 3
23) -44 -23 3
24) 25 -43 5
25) -44 22 35
26) 29 -31 27
27) 40 31 11
28) 14 50 -40
29) 50 43 -34
30) -7 49 3
31) -10 46 -8
32) -32 -20 46
33) -31 8 47
34) 27 40 -5
35) -46 -16 11
36) 13 22 19
37) 6 -12 -48
38) 38 24 -33
39) 14 -41 -1
40) -19 4 -47
41) 5 -26 -24
42) -43 42 -20
43) -43 34 26
44) 7 31 21
45) -17 -25 -27
46) -11 2 35
47) -8 40 44
48) 44 37 25
49) -43 -18 -16
50) -17 33 47
51) -2 44 47
52) -5 -11 -44
53) -19 -5 -20
54) -10 16 42
55) -16 17 43
56) -1 34 25
57) 13 45 -12
58) 7 3 -14
59) -48 14 50
60) -39 42 -40
61) -36 -18 -33
62) 37 49 19
63) 50 35 -26
64) -11 -1 39
65) 14 23 4
66) -30 37 -10
67) -18 50 -37
68) -10 -48 -12
69) -15 18 -23
70) 46 39 41
71) -17 31 -32
72) -24 -39 -2
73) -30 -18 -36
74) 17 -15 27
75) 38 23 -13
76) -24 34 9
77) -6 -16 19
78) 47 -34 19
79) -11 29 37
80) 35 17 38
81) -44 -34 -22
82) -30 -15 -24
83) 14 32 -13
84) -14 37 -13
85) -18 -28 -39
86) -5 26 -7
87) 20 -23 -32
88) 30 -11 -28
89) 1 -11 -16
90) -19 -18 -23
91) 41 -8 -29
92) -28 -33 -42
93) -34 38 -42

94) -45 -3 -26
95) -31 -34 5
96) 29 -17 -30
97) -9 2 -34
98) 46 23 -37
99) -39 -33 27
100) -46 -24 40
101) -2 48 -43
102) -44 -28 33
103) -5 49 30
104) -48 -23 49
105) 23 -40 43
106) -24 -4 2
107) 27 -34 -19
108) 3 -4 14
109) -28 -21 -4
110) 30 47 7
111) -40 4 -36
112) -9 -21 38
113) 50 -41 -11
114) 43 11 34
115) -5 -16 -36
116) -18 44 37
117) 36 20 25
118) -36 -24 -21
119) 22 -18 30
120) -49 9 12
121) -44 -18 25
122) -46 -34 -19
123) -39 -48 -31
124) -6 -10 2
125) -2 -21 30
126) -27 50 -21
127) -48 -2 -47
128) 7 3 45
129) -15 9 14
130) 27 -3 6
131) -45 -47 13
132) -34 23 -41
133) -14 -18 -9
134) 46 -7 38
135) -6 -48 4
136) -35 -33 43
137) -5 -43 9
138) -4 40 18
139) -37 -20 -5
140) -33 -10 -40
141) 14 -17 -18
142) 3 -47 18
143) 46 -18 20
144) 47 22 -16
145) 8 -1 39
146) 48 18 -36
147) 21 48 -22
148) -2 -13 -26
149) 14 -36 -7
150) -30 -20 47
151) 18 -37 44
152) 10 33 31
153) 10 41 47
154) 24 -25 -39
155) 44 -36 45
156) 10 -5 -6
157) -48 -27 26
158) 26 30 -7
159) -10 7 24
160) 1 6 -42
161) 3 2 16
162) 32 -28 -9
163) -17 35 30
164) -23 15 30
165) 50 30 2
166) -47 -15 27
167) -27 -41 -29
168) 15 27 43
169) -29 -34 -19
170) -43 -19 -37
171) 19 16 -45
172) -34 2 41
173) -35 -17 37
174) -47 48 -37
175) 47 -12 -21
176) 35 -13 -15
177) -35 -45 -13
178) -15 37 5
179) -17 30 16
180) -45 -49 -23
181) -1 -45 -48
182) -31 33 -47
183) -9 14 25
184) -35 44 33
185) 29 31 -14
186) 14 43 -11
187) 50 21 10
188) 40 -43 6
189) -29 -18 -43
190) 23 33 10
191) 11 -41 -19
192) -34 -40 44
193) 47 -31 10
194) 11 15 18
195) -23 4 14
196) -34 -30 9
197) 10 -38 41
198) 5 -34 -11
199) -48 24 23
200) 5 7 50
201) -27 -28 -44
202) 34 -12 30
203) 31 12 -23
204) 31 -35 40
205) -2 -38 -50
206) -19 -5 -43
207) -47 30 10
208) 16 -50 -24
209) -14 26 -25
210) 7 -30 -40
211) 9 49 -24
212) -23 -42 -17
213) 6 -8 -15
214) -39 -28 -43
215) -25 -49 42

2. Results

Results after execution of truth assignment algorithm

- Result for dataset-1: 3-variables 4-clauses

The Formula is Satisfiable: Satisfying truth-value assignments are:

x	-3	0
x	-2	1
x	-1	1
x	0	0
x	1	0
x	2	0
x	3	1

- Result for dataset-2: 3-variables 8-clauses

The Formula is not Satisfiable : Satisfying truth-value assignments are:

x	-3	1
x	-2	1
x	-1	1
x	0	0
x	1	1
x	2	1
x	3	1

- Result for dataset-3: 10-variables 43-clauses

The Formula is Satisfiable: Satisfying truth-value assignments are:

x	-10	1	x	-3	0	x	4	1
x	-9	1	x	-2	1	x	5	1
x	-8	0	x	-1	0	x	6	0
x	-7	0	x	0	0	x	7	1
x	-6	1	x	1	1	x	8	1
x	-5	0	x	2	0	x	9	0
x	-4	0	x	3	1	x	10	0

- Result for dataset-4: 20-variables 86-clauses

The Formula is Satisfiable: Satisfying truth-value assignments are:

x	-20	0	x	-13	1	x	-6	0
x	-19	1	x	-12	0	x	-5	1
x	-18	0	x	-11	0	x	-4	0
x	-17	1	x	-10	0	x	-3	1
x	-16	0	x	-9	1	x	-2	1
x	-15	0	x	-8	1	x	-1	1
x	-14	1	x	-7	0	x	0	0

x	1	0	x	8	0	x	15	1
x	2	0	x	9	0	x	16	1
x	3	0	x	10	1	x	17	0
x	4	1	x	11	1	x	18	1
x	5	0	x	12	1	x	19	0
x	6	1	x	13	0	x	20	1
x	7	1	x	14	0			

- Result for dataset-5: 20-variables 91-clauses

The Formula is Satisfiable: Satisfying truth-value assignments are:

x	-20	0	x	-6	0	x	8	0
x	-19	1	x	-5	1	x	9	1
x	-18	1	x	-4	1	x	10	0
x	-17	0	x	-3	1	x	11	0
x	-16	1	x	-2	1	x	12	0
x	-15	0	x	-1	0	x	13	0
x	-14	0	x	0	0	x	14	1
x	-13	1	x	1	1	x	15	1
x	-12	1	x	2	0	x	16	0
x	-11	1	x	3	0	x	17	1
x	-10	1	x	4	0	x	18	0
x	-9	0	x	5	0	x	19	0
x	-8	1	x	6	1	x	20	1
x	-7	1	x	7	0			

- Result for dataset-6: 40-variables 129-clauses

The Formula is Satisfiable: Satisfying truth-value assignments are:

x	-40	1	x	-27	1	x	-14	0
x	-39	1	x	-26	1	x	-13	0
x	-38	1	x	-25	1	x	-12	1
x	-37	1	x	-24	1	x	-11	0
x	-36	1	x	-23	1	x	-10	0
x	-35	1	x	-22	1	x	-9	0
x	-34	1	x	-21	1	x	-8	0
x	-33	1	x	-20	0	x	-7	1
x	-32	1	x	-19	0	x	-6	0
x	-31	1	x	-18	0	x	-5	0
x	-30	1	x	-17	1	x	-4	0
x	-29	0	x	-16	0	x	-3	0
x	-28	0	x	-15	1	x	-2	0

x	-1	1	x	13	1	x	27	0
x	0	0	x	14	1	x	28	1
x	1	0	x	15	0	x	29	1
x	2	1	x	16	1	x	30	0
x	3	1	x	17	0	x	31	0
x	4	1	x	18	1	x	32	0
x	5	1	x	19	1	x	33	0
x	6	1	x	20	1	x	34	0
x	7	0	x	21	0	x	35	0
x	8	1	x	22	0	x	36	0
x	9	1	x	23	0	x	37	0
x	10	1	x	24	0	x	38	0
x	11	1	x	25	0	x	39	0
x	12	0	x	26	0	x	40	0

- Result for dataset-7: 40-variables 172-clauses

The Formula is Satisfiable: Satisfying truth-value assignments are:

x	-40	0	x	-16	1	x	8	0
x	-39	1	x	-15	0	x	9	1
x	-38	1	x	-14	1	x	10	0
x	-37	1	x	-13	0	x	11	0
x	-36	1	x	-12	0	x	12	1
x	-35	1	x	-11	1	x	13	1
x	-34	0	x	-10	1	x	14	0
x	-33	0	x	-9	0	x	15	1
x	-32	1	x	-8	1	x	16	0
x	-31	0	x	-7	0	x	17	0
x	-30	0	x	-6	1	x	18	1
x	-29	0	x	-5	1	x	19	0
x	-28	1	x	-4	0	x	20	1
x	-27	1	x	-3	0	x	21	0
x	-26	0	x	-2	1	x	22	1
x	-25	1	x	-1	0	x	23	0
x	-24	0	x	0	0	x	24	1
x	-23	1	x	1	1	x	25	0
x	-22	0	x	2	0	x	26	1
x	-21	1	x	3	1	x	27	0
x	-20	0	x	4	1	x	28	0
x	-19	1	x	5	0	x	29	1
x	-18	0	x	6	0	x	30	1
x	-17	1	x	7	1	x	31	1

x	32	0	x	35	0	x	38	0
x	33	1	x	36	0	x	39	0
x	34	1	x	37	0	x	40	1

- Result for dataset-8: 50-variables 215-clauses

The Formula is Satisfiable: Satisfying truth-value assignments are:

x	-50	1	x	-16	1	x	18	0
x	-49	0	x	-15	1	x	19	1
x	-48	1	x	-14	0	x	20	1
x	-47	1	x	-13	1	x	21	0
x	-46	0	x	-12	1	x	22	0
x	-45	1	x	-11	0	x	23	0
x	-44	0	x	-10	0	x	24	0
x	-43	1	x	-9	0	x	25	0
x	-42	0	x	-8	0	x	26	1
x	-41	1	x	-7	0	x	27	1
x	-40	1	x	-6	0	x	28	0
x	-39	1	x	-5	1	x	29	0
x	-38	0	x	-4	1	x	30	0
x	-37	0	x	-3	1	x	31	1
x	-36	1	x	-2	0	x	32	0
x	-35	0	x	-1	1	x	33	0
x	-34	1	x	0	0	x	34	0
x	-33	1	x	1	0	x	35	1
x	-32	1	x	2	1	x	36	0
x	-31	0	x	3	0	x	37	1
x	-30	1	x	4	0	x	38	1
x	-29	1	x	5	0	x	39	0
x	-28	1	x	6	1	x	40	0
x	-27	0	x	7	1	x	41	0
x	-26	0	x	8	1	x	42	1
x	-25	1	x	9	1	x	43	0
x	-24	1	x	10	1	x	44	1
x	-23	1	x	11	1	x	45	0
x	-22	1	x	12	0	x	46	1
x	-21	1	x	13	0	x	47	0
x	-20	0	x	14	1	x	48	0
x	-19	0	x	15	0	x	49	1
x	-18	1	x	16	0	x	50	0
x	-17	1	x	17	0			