F-019

BOINC による SAT ソルバーの並列実行

Parallelizing SAT Solver with BOINC

力 規晃† 越村 三幸‡ 藤田 博‡ 長谷川 隆三‡
Noriaki Chikara Miyuki Koshimura Hiroshi Fujita Ryuzo Hasegawa

1．はじめに

近年，高速な充足可能性判定問題を解くためのソルバー（SAT ソルバー）の登場により，回路やソフトウェアのシステム検証[1][5]，プランニング問題やスケジューリング問題[4]，制約充足問題，制約最適化問題[3]，定理証明などの問題を命題論理の充足可能性判定問題(SAT 問題)に変換し解く問題解決手法が開発され，大きな性能の向上をもたらしている[2][6].

しかし，まだ，求解できない問題も数多くある．このような難解な問題を解く一つの方法として，複数のコンピュータで並列に解く方法がある．

本研究では計算資源をボランティアに提供してもらうことを特徴とする BOINC というグリッドコンピューティングプラットホームを用いて，SAT ソルバーMiniSAT を並列実行するシステムを構築した．そして，性能の検証のため，巨大なナンバープレースを対象として並列化の評価実験を行い，考察する．

2．SAT

SAT とは命題論理の充足可能性判定(Boolean Satisfiability(testing);SAT)のことである．

2.1 SAT問題

SAT 問題とは命題論理の充足可能性判定問題のことであり，連言標準形（CNF）で表される命題論理式が与えられたとき，全体の値を真にすることが可能(充足可能)かを判定する問題である．CNF の例を式(1)に示す．

$$(s_1 \vee s_2) \wedge (s_1 \vee \neg s_2) \wedge (s_3 \vee s_4) \wedge (\neg s_3 \vee \neg s_4) \wedge (s_1 \vee \neg s_3) \wedge (\neg s_2 \vee \neg s_4) \quad (1)$$

CNF では変数または変数の否定をリテラルと呼ぶ，また，リテラルを論理和∨で結合したものを節と呼ぶ．式(1)ではそれぞれの括弧の部分が節である．また，リテラルを一つしか持たない節を単位節と呼ぶ．

式(1)の例では s_1, s_2 及び s_3 には真，s_4 には偽を割り当てると，式(1)は真となり，充足可能となる．充足可能であることを SAT と呼び，充足不可能であることを UNSAT と呼ぶ．また，充足可能である場合，その変数割当をモデル(Model)と呼ぶ．

2.2 SAT ソルバー

SAT 問題を解くソルバー のことを SAT ソルバーという．

多くの SAT ソルバーでは充足可能性を判定した結果が充足可能である場合，モデルが得られる．

本研究では既存の高速な SAT ソルバーである MiniSat[6][7][10]を用いる．なお，MiniSat とは，オープンソースの SAT ソルバーである．リテラル監視による高速な伝搬と後戻り，衝突節の学習による探索空間の枝刈り，バックジャンピング（衝突解析による後戻り），決定変数選択のヒューリスティックの導入等により，高速な探索を実現しており，SAT Competitions[11] で優秀な成績を収めている．

MiniSAT の入力として与える SAT 問題の形式は DIMACS CNF 形式である．DIMACS CNF 形式の例を図 1 に示す．

```
p cnf 7 3
3 -7 -2 0
-6 0
1 -3 4 -2 0
```

図 1 DIMACS CNF 形式

DIMACS CNF 形式は，それぞれの行が節を表す．変数を番号で表し，OR を空白，NOT をマイナスで表すものである．節の終わりには 0 を置く．また 1 行目には変数の個数，節の数を記述する．

2.3 SATソルバーを用いた問題解決

一般的に実際の問題を SAT ソルバーで解く手順は図 2 のように行う[2]．まず，もとの問題を SAT 符号化(SAT encoding)を行い，SAT 問題(SAT Problem)に変換する．この SAT 問題を，SAT ソルバーを用いて解き，SAT もしくは UNSAT を得る．SAT の場合，モデルが得られるので，これを復号化(decoding)し，もとの問題の解を得る．

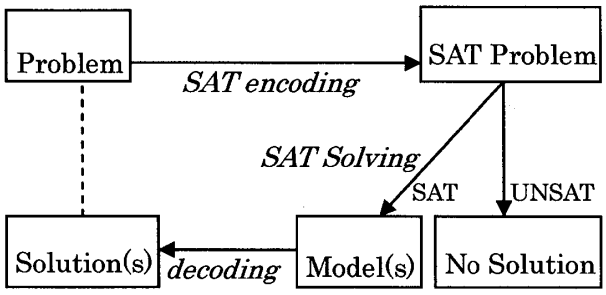

図 2 SAT ソルバーを用いた問題解決

3. BOINC

BOINC[12][14]とはカリフォルニア大学バークレー校で開発されたグリッドコンピューティングプラットホームであり，クライアントサーバー型で分散並列処理を行う．計算資源であるクライアント PC をボランティアに提供してもらい，インターネット接続された一般の PC を用いることが特徴である．特に SETI[13]という地球外知的生命体の

† 徳山工業高等専門学校, Tokuyama College of Technology
‡ 九州大学, Kyushu University

探査のプロジェクトで，電波望遠鏡で受信した電波の解析に用いられていることで良く知られている．

4. SAT 問題の分割

難解な問題を高速に解く手段の一つとして，複数台の PC で並列に処理をする手段がある．これを実現させるため，一つの問題を分割していく必要がある．

本研究では，ある 1 つの着目した命題変数を真と偽でそれぞれ確定させることで問題をそれぞれ作成し，2 分割する．実際には，ある変数の真偽を確定させるため，その変数を含む単位節をもとの問題に追加する．

このようにして問題の 2 分割が可能であるが，単純に 1 度だけ 2 分割するだけでは解けない問題もあることが予想される．このため本研究では，まずクライアントで問題を解かせ，一定時間で解けないでタイムアウトした場合，その問題をある変数に着目して 2 分割し，再度それぞれの問題をクライアントで解く，着目する変数を次々と変えながら，これを繰り返すことにより最終的にもとの SAT 問題を解く．また，単位節などにより，着目する変数の真偽がすぐに判定できる場合がある．このような変数は，分割の際に着目する変数には選択しないようにする．

5. システム構成

本研究で用いるシステムの構成と流れを図 3 に示す．

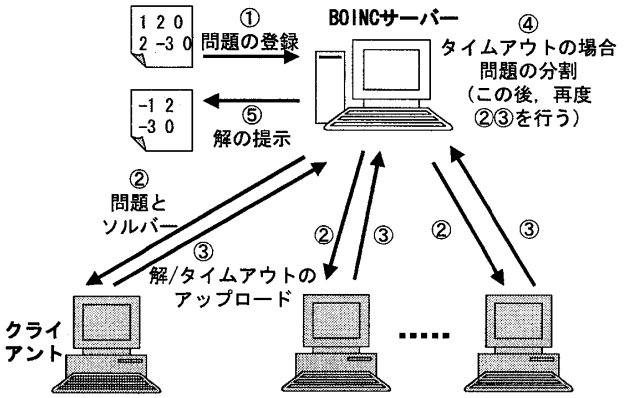

図 3 システム構成と流れ

クライアントは事前に BOINC クライアントのソフトウェアを動作できるようにしておき，BOINC サーバーに登録しておく．①BOINC サーバーに SAT 問題を登録する．②問題とソルバーをクライアントにダウンロードし，クライアントはそのダウンロードされた問題とソルバーを用いて問題を解く．③クライアントで問題が解けたか，もしくはタイムアウトした場合，結果を BOINC サーバーにアップロードする．④クライアントで解いた結果がタイムアウトであった場合，問題を分割し，再度②③を行う．⑤SAT が一つでも得られた場合，もしくは分割した全ての問題が UNSAT だった場合，結果が得られたとして解の提示を行う．

6. 実験

本研究の並列化手法とシステムの検証のため，実験を行った．ここで，対象とする問題は巨大ナンバープレースを SAT 問題に変換したものである．

6.1 ナンバープレース

よく知られているナンバープレースは，3×3 のブロックに区切られた 9×9 正方形のマスに 1～9 の数字をいれるペンシルパズルのひとつである[15]．ルールは，空いているマスに 1～9 の数字をいれること，各行，各列およびブロック内に同じ数字が重複してはならないことである．また，問題には初めからヒントとしていくつかの数字が与えられてある．一般的には様々なサイズのナンバープレースが存在する．図 4 にナンバープレースの問題例とその正解を示す．

1				5				6
	9	3		1		8	4	
	5		6		8		2	
		5	1		4	6		
2	6			3			5	4
		1	7		5	9		
	4		3		1		6	
	7	6		8		4	1	
8				4				7

(a)問題

1	2	8	4	5	3	7	9	6
6	9	3	2	1	7	8	4	5
7	5	4	6	9	8	3	2	1
9	8	5	1	2	4	6	7	3
2	6	7	8	3	9	1	5	4
4	3	1	7	6	5	9	8	2
5	4	9	3	7	1	2	6	8
3	7	6	5	8	2	4	1	9
8	1	2	9	4	6	5	3	7

(b)正解

図 4 ナンバープレースの例

6.2 ナンバープレースをSATソルバーで解く

ナンバープレースを SAT ソルバーで解く場合，一般的に多くの問題を SAT ソルバーで解く場合と同じように，図 5 のような手順で解く．まずナンバープレースの問題を SAT 符号化し，DIMACS CNF 形式の SAT 問題に変換する．その SAT 問題を SAT ソルバーに入力として与えて解く．そうすると，充足可能であった場合，SAT の形式のモデルが得られる．このモデルを SAT 復号化し，ナンバープレースの解の配置が得られる．

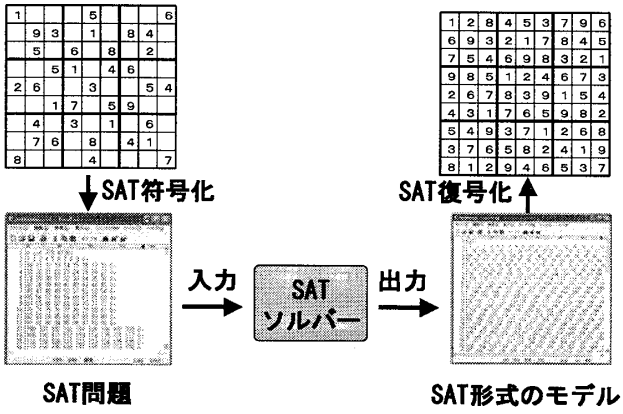

図 5 ナンバープレースを SAT ソルバーで解く手順

6.3 ナンバープレースのSAT問題への変換

ナンバープレースを SAT 符号化し SAT 問題に変換して解く試みは多数ある[8][9]．本研究では，まず，ナンバープレースのルールから図 6 のような制約を考える．

(a) 各マスに 1～N のいずれかの数字が入る
(b) 各マスに 1～N の数字が 2 つ以上入らない
(c) 各行に同じ数字が 2 回以上現れない
(d) 各列に同じ数字が 2 回以上現れない
(e) 各ブロックに同じ数字が 2 回以上現れない
(f) 最初から数字が入っているマスはその数字になる
（ここで，N はナンバープレースの一辺のサイズである）

図 6 ナンバープレースの制約

これを CNF 形式に変換すると(a)～(f)のようになる．ただし，ここで，$s_{x,y,z}$ は座標(x,y)のマスに，z の数字を当てはめることを示す変数である．

(a) 各マスに 1～N のいずれかの数字が入る

$$\wedge_{x=1}^{N} \wedge_{y=1}^{N} \vee_{z=1}^{N} s_{x,y,z} \tag{2}$$

(b) 各マスに 1～N の数字が 2 つ以上入らない

$$\wedge_{x=1}^{N} \wedge_{y=1}^{N} \wedge_{z=1}^{N-1} \wedge_{i=z+1}^{N} \left(\overline{s_{x,y,z}} \vee \overline{s_{x,y,i}}\right) \tag{3}$$

(c) 各行に同じ数字が 2 回以上現れない

$$\wedge_{y=1}^{N} \wedge_{z=1}^{N} \wedge_{x=1}^{N-1} \wedge_{i=x+1}^{N} \left(\overline{s_{x,y,z}} \vee \overline{s_{i,y,z}}\right) \tag{4}$$

(d) 各列に同じ数字が 2 回以上現れない

$$\wedge_{x=1}^{N} \wedge_{z=1}^{N} \wedge_{y=1}^{N-1} \wedge_{i=y+1}^{N} \left(\overline{s_{x,y,z}} \vee \overline{s_{x,i,z}}\right) \tag{5}$$

(e) 各ブロックに同じ数字が 2 回以上現れない

$$\wedge_{z=1}^{N} \wedge_{i=0}^{\sqrt{N}-1} \wedge_{j=0}^{\sqrt{N}-1} \wedge_{x=1}^{\sqrt{N}} \wedge_{y=1}^{\sqrt{N}} \wedge_{l=1}^{\sqrt{N}} \wedge_{k=1}^{\sqrt{N}} \left(\overline{s_{(\sqrt{N}i+x),(\sqrt{N}j+y),z}} \vee \overline{s_{(\sqrt{N}i+l),(\sqrt{N}j+k),z}}\right) \tag{6}$$

ただし，$(\sqrt{N}i+x,\sqrt{N}j+y)=(\sqrt{N}i+l,\sqrt{N}j+k)$ となる節は除く．

(f) 最初から数字が入っているマスはその数字になる

$$\wedge_{(x_C,y_C,z_C)} s_{x_C,y_C,z_C} \tag{7}$$

実際の DIMACS CNF 形式では一つの変数は複数の添え字を持てないため，$s_{x,y,z}$ の変数番号は次の式となる．

$$yN^2 + xN + z \tag{8}$$

ここで，N はナンバープレースの一辺のサイズ，(x,y) はナンバープレースのマスの座標，z はそのマスに割り当てた数字である．

6.4 実験の条件

本実験では BOINC サーバー1 台とクライアント 1 台を用いて問題を分割して模擬的に分散並列で解き，分割せずに解いた場合との比較を行った．

今回用いた問題は，分割せずに解いた場合 10 分～20 分で解ける 25×25 のナンバープレース 5 問であり，全て正解が存在する．つまり，充足可能となる問題である．

また，タイムアウトの時間は 3 分とした．この時間を超えた場合，ある変数に着目し問題を 2 分割する．この着目する変数は単位節などで真偽がすぐわかる変数を除いて，変数番号順で機械的に選んだ．

6.5 実験結果

実験結果を表 1 に示す．ここで，分割回数は解が得られた分割問題が何度分割して生成されたものかを示す．分割回数を n とすると，もとの問題がいくつの問題に分割されたかは 2^n であると推測できる．また，本実験は擬似的な並列処理であるため，分割ありの時間は次の式で得られたものである．

$$N_d \times T_{timeout} + T_{ds} \tag{9}$$

ここで，N_d は分割数，$T_{timeout}$ はタイムアウト時間，T_{ds} は解が得た分割問題を解くのに要した時間である．

表 1 実験結果

問題	分割なし① (min)	分割回数	分割あり② (min)	②／① (%)
問題 1	17.2	9	29.8	174
問題 2	17.0	3	14.4	84
問題 3	16.5	1	4.5	28
問題 4	11.0	1	3.7	33
問題 5	10.1	1	5.8	57

7. 考察

実験結果より，5 問中 4 問で分割なしの場合よりも速く問題を解くことができた．また，通常，並列処理の台数効果により，n 台に分割して処理した場合，1/n の時間で処理が終わるのが，理想的な状態である．しかし，問題 3,4 では 1 回分割しているので，通常 50%の時間で解けるのが理想的な状態であるが，これを越える速さで解けている．一方，問題 1 は極端によくない結果となったのが，これは，タイムアウト時間が適切でなく，解ける直前にタイムアウトになり，分割した問題を最初から解きなおすことを繰り返しているものと思われる．このため，タイムアウト時間の調整が必要だと思われる．

8. おわりに

本論文では BOINC というグリッドコンピューティングプラットホームを用いて，SAT ソルバーMiniSAT を並列実行するシステムを構築し．そして，性能の検証のため，巨大なナンバープレースを対象とし実験と考察を行った．今後の課題として，問題を解く際のタイムアウト時間をチュ

ーニングする方法の確立が必要だと思われる．また，今回は 5 つのナンバープレース問題でしか実験をしていない．このため，より多くの問題で検証をすることが必要である．ナンバープレース以外の問題にも今回の手法を試し，検証を行っていきたい．

謝辞

本研究は科研費(20240003)の助成を受けたものである．

参考文献

[1] 藤田昌宏, SAT アルゴリズムの最新動向, 電子情報通信学会誌，Vol.90,No.12,pp.1067-1072, 2007

[2] 井上克己,田村直之,SAT ソルバーの基礎, 人工知能学会誌，Vol.25,No.1,pp.57-67, 2010

[3] 田村直之,丹生智也,番原睦則, 制約最適化問題と SAT 符号化, 人工知能学会誌，Vol.25,No.1,pp.77-85, 2010

[4] 鍋島英知,SAT によるプランニングとスケジューリング, 人工知能学会誌，Vol.25,No.1,pp.114-121, 2010

[5] 番原睦則,田村直之, SAT によるシステム検証, 人工知能学会誌，Vol.25,No.1,pp.122-129, 2010

[6] 鍋島英知,宋剛秀,高速 SAT ソルバーの原理, 人工知能学会誌，Vol.25,No.1,pp.68-76, 2010

[7] N.Eén, N.Sörensson, An extensible SAT-solver, Proc. 6th Int. Conference on Theory and Applications of Satisfiability Testing (SAT2003),pp.502-518, 2003

[8] 田中哲朗, パズルにまつわる最近の話題, 人工知能学会誌，Vol.23,No.3,pp.367-372, 2010

[9] I.Lynce, J.Ouaknine, Sudoku as a SAT Problem, Proc. 9th Int. Symposium on Artificial Intelligence and Mathematics (AIMATH2006), 2006

[10] MiinSat Page,http://minisat.se/

[11] SAT Competitions, http://www.satcompetition.org/

[12] BOINC,http://boinc.berkeley.edu/

[13] SETI@home,http://setiatohome.berkeley.edu/

[14] 岩崎正剛,オープンソースのグリッドエンジン BOINC 入門,Linux ソフトウェアアンテナ,技術評論社,pp.150-159,2005

[15] 棚床弘樹,鉛筆パズルゲームプログラミング ナンバープレース・お絵かきパズル・ナンバークロスワードのアルゴリズム,ソフトバンククリエイティブ,2007