社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－HPC－99 (5)
2004／7／30

SATソルバzchaffのMPIによる並列化

大橋智昭† 稲垣良一† 上田和紀†

本研究では高性能SATソルバzchaffをクラスタ上で効率的に並列化する手法を提案している．近年のSATソルバは変数の衝突を学習し，節として追加する事で同じ変数の衝突を起こさないようする事で，SATの探索域を大幅に削減する．本研究の手法では各ノードがそれぞれランダムに決定した探索域の探索を行い，探索の途中でそれぞれのノードが学習した結果の中で探索域削減効果の高いと思われる節を交換することで効率的な並列化を実現している．本研究で作成した並列SATソルバMPI-zchaffは，充足可能な問題の解の発見において高い性能向上を実現し，SAT Competiton 2003のunsolved problemの解を発見できた．

Parallelization of the SAT solver zchaff with MPI

Tomoaki OOHASHI,† Ryoichi INAGAKI† and Kazunori UEDA†

We propose an efficient parallelization method of the high-performance SAT solver zchaff. Nowadays, most SAT solvers learn conflicts of variables and by avoiding the same conflicts of variables reduce the search space. Our method has achieved effective parallelization by exchanging lemma each computation node learned in searching its own search space. We have implemented a parallel SAT solver by using zchaff and MPI and accelerated discovery of solutions. In particular, we discovered a solution of an unsolved problem in SAT Competition 2003.

1. はじめに

充足可能性問題 (Boolean Satisfiability Problem, 以下SATと呼ぶ) は典型的なNP問題として計算機の分野において理論的に強い興味の対象となってきた．また様々の実アプリケーション (プランニング，ソフトウェア検証，ハードウェア検証，定理証明など) から発生した問題がSATに定式化可能でありSATを高速に解く意義はとても大きいと言える．

このような背景から多数のSATソルバが考案され改良が繰り返されてきた．近年のSATアルゴリズムの大きな特徴に衝突学習である．解の探索中に変数が衝突を起こした際，その衝突の原因を解析し，節として加える事で同じような探索域の探索を避ける事ができる．これによりハードウェア検証など，問題中にある種の対称性がある実アプリケーションから発生したSATの解の探索速度が飛躍的に高まり，何千～何万変数の問題の解の判定が可能になった．

SATソルバzchaffはSATの性能を競う大会であるSAT competition 2002でindustrial, handmadeの2部門で最優秀のソルバになった非常に高性能なソルバである．しかしながらzchaffなどの高性能なソルバを用いても適当な時間内に解けない問題はいまだに多数存在する．

本研究ではzchaffをソルバコアに用いて，クラスタ上で並列化する事により高速化を目指した．SATの探索域はその大きさがあらかじめ予測できない事，対称性のある探索域が重複して探索されてしまう事などの問題がある．そこで本研究では各ノードがSATの探索域をランダムに決定して探索し，その学習節をノード間で相互に通信し合う事で効率的に解の探索を行った．本研究で実装した並列SATソルバをMPI-zchaffと呼ぶ．

以下2章でSATについて述べ，3章でzchaffを初めとするSATソルバの探索アルゴリズムの概要について述べ，4章では並列化の問題点とMPI-zchaffの実装について述べる．5章で性能評価結果を示し，6章で今後の課題とまとめを述べる．

2. SAT

SATソルバは，論理積標準形 (Conjunctive Normal Form, 以下CNF) に形式化した論理式の充足可能性を探索によって判定する．CNF形式は探索域の削減に効果が高く，しかもどのようなSATの問題も

† 早稲田大学大学院 理工学研究科 情報・ネットワーク専攻

多項式時間内に CNF 形式に変換できる.

CNF では変数と変数に否定がついたものをリテラルと言い，リテラルを論理和でつなげたものを節 (以下 clause) という．clause を論理積でつなげたものが全体の式となる．SAT は全ての変数に真偽値を割り当て，全体の式が充足するか，またはどのような割り当てをしても全体の式が充足しないか判定する問題である．

SAT の例

式 $= (a + b')(b + c)(a' + c')(a + b' + c)$

※ $a = True$, $b = True$, $c = False$ で全体の式は充足する．

3. SAT 探索アルゴリズム

SAT の探索アルゴリズムは，体系的アルゴリズム (complete algorithm) と確率的アルゴリズム (stochastic algorithm) に大別できる．体系的アルゴリズムは解を見つけるか，解がない事を示すまで探索を続ける．一方確率的アルゴリズムは解の発見は非常に高速だが，充足不可能な事を示せないため，プランニングや FPGA routing など充足不可能なことを示す事が第一義でない類の問題では非常に有効である．しかしながら，ほとんどの問題では充足不可能な事を示すのが重要である．以下に zchaff などの体系的探索を行うソルバで採用されているアルゴリズムを述べる．

3.1 DPLL

近年の zchaff などのソルバは DPLL (Davis-Putnam-Loveland-Logemann) アルゴリズムをフレームワークとしている．その概要を図 1 に示す．関数の意味はそれぞれ以下の通りである．

- 関数 `Decide()` は次の割り当ての変数をヒューリスティックスなどを元に決定する．もし次に割り当てる変数が無いとき，つまり全ての変数に真偽値が割り当てられたならばその問題は充足可能である．
- 関数 `Deduce()` は現在の真偽値の割り当てから，演繹的に真偽値が決定できる変数を見つけ出し割り当てを行う．ある clause の全てのリテラルが偽と割り振られた時を衝突を起こしているという．Deduce() 関数は衝突を起こした時に FALSE を返す．
- 関数 `Analyze_Conflict()` は衝突を解析し，衝突が起きていない状態にバックトラックさせる．もしバックトラックできない状態になったらその問題は充足不可能である．

```
while (true){
    if (Decide()){
      while (!Deduce()){
         if (!Analyze_Conflict())
                    return UNSAT;
      }
    }
    else{
        return SAT;
    }
}
```

図 1 ループ版 DPLL

3.2 Deduce アルゴリズム

DPLL の中の Deduce アルゴリズムでどのソルバでも必ず使われているものは単位節規則 (unit clause rule) である．ある clause が一つの未割り当てのリテラルを除き，他の全てのリテラルが偽となるように真偽値が割り振られている状態を unit clause という．残った一つのリテラルが真となるように真偽値を与えないと，全体の式が真になりえないので未割り当ての変数の真偽値を決定することができる．

zchaff をはじめとする近年の SAT ソルバは，実行時間の約 90% 以上を unit clause rule の実行に使っている．

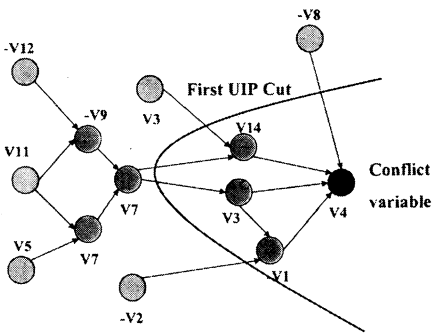

図 2 implication graph

3.3 Conflict Analysis & Learning

変数に割り当てを進めた結果として，衝突が起きた時，ソルバは衝突を起こしている原因を分析し，特定の探索空間には解がないことを判断して新たな探索

空間を探索をする．また学習の結果は学習節 (learn clause や lemma と呼ぶ) として追加されて同じ衝突を起こさないようになる．

衝突の学習方法は図 2 のような implication graph におけるグラフカット問題に相当する．図 2 は V11 がヒューリスティクスにより決められた変数で，矢印が unit clause rule の伝播を示し，V4 が衝突を起こしている変数である．いくつかの学習方法 (≒グラフのカット方法) が考えられるが，参考文献 3) では UIP(Unique Implication Points: 決定変数から衝突変数までに必ず通る変数, 決定変数自身も入る) の中で，衝突変数側から見た初めての UIP である FirstUIP でのカット方法が平均して効率的な探索域削減効果を示し，高速であるとしている．図 2 においての FirstUIP カットによる得られる lemma は (V7' + V3' + V2 + V8) である．lemma は元の式に冗長な clause を追加するため，探索域の削減の効果と lemma を加える事により問題サイズが大きくなり探索プロセスが重くなる事のトレードオフとなる．

3.4 Decision アルゴリズム

unit clause rule による真偽の割り当てが不可能になった時点で，真偽値が未割り当ての変数の中から真偽値を割り振る変数を何らかのヒューリスティクスで選ばなければならない．良い decision ヒューリスティクスは探索の分岐回数を減らせるので大幅に探索時間を削減できる．代表的なヒューリスティクスに以下のようなものがある．

- DLIS (dynamic largest combined sum)
  現在未割り当ての変数の中から一番現在の状態に多く出現する変数を選択する方法．各変数ごとにカウンタを持たせてやればよいので実装は簡単にもかかわらず高い．しかしながら新たな割り当てやバックトラックが発生した時にカウンタの値を更新しなければならない．
- VSIDS (variable state independent decaying sum)
  衝突学習機能に合わせた変数選択方法．各変数のカウンタの初期値を変数の最初の出現頻度とし，その後衝突が起きる時に追加した lemma 内の変数のカウンタを増やす．またカウンタの値を定期的に減らす事で VSIDS は「最近衝突を起こしている変数」の重要度を表す．VSIDS は探索全体の実行時間に対する VSIDS の実行時間が DLIS に比べて非常に小さいにもかかわらず良い変数選択ができる．

4. 並列化の問題点と実装

4.1 ソルバコア

本研究の並列 SAT ソルバのソルバコアには zchaff を採用する．最大の理由に並列 SAT ソルバの性能はソルバコアに大きく依存してしまうのでなるべく高速なソルバを使用したかったためである．zchaff は SAT Competiton 2002 で industrial, handmade の 2 部門で最優秀であったソルバでありかつソースコードが参考文献 4) の URL 上で入手可能である．以下に関連研究であるいくつかの並列 SAT ソルバとそのソルバコアについて解説する．

- //Satz[6] はソルバコアに satz を使用している．探索域の広さを変数の割り当て個数から大雑把に推測し動的な負荷分散を可能にしている．satz は変数の衝突学習機能はなく，変数の選択方法に DLIS を採用しており zchaff と比べるとほとんどの問題において数段遅い．
- WalkSAT-MPI[7] はソルバコアに walksat を使用している．walksat は確率的アルゴリズムによる探索を行うソルバで，各ノードがランダムアルゴリズムで使用する種をそれぞれずらす事で並列探索する．
- multisat[9] は PrologCafe をプラットフォームにしてマルチスレッド化することにより, Java で記述された異なる SAT ソルバを実行する．multisat は Java で記述された SAT ソルバならインターフェイスをあわせるだけで並列実行できる．この実験的ソルバでは satz, walksat, zchaff と単純に割り当てをするソルバの 4 つのプログラムを同時に並列実行する．複数のソルバを並列実行することで互いのソルバの得意，不得意な問題を補い，結果として個々のソルバのパラメタを変えただけのソルバより平均的に早いソルバを目指している．実験において充足性の判定をしたソルバのほとんどが zchaff と walksat であった．
- PaSAT[10] は zchaff の前身である chaff で並列 SAT ソルバを実装しており，lemma の交換による superlinear な性能向上の探索の可能性を示している．

4.2 並列化の問題点

SAT の探索域を前章で述べたアルゴリズムを用いて並列探索をする時に考慮する事として，

- unit clause rule が何個の変数の真偽値を，割り当てるか事前にわからないため，探索木の大きさが予測できず負荷分散が難しい．

- あるノードがこれから探索する探索域は既に別のノードの衝突学習プロセスにより解が無いことがわかっているかもしれない.

が挙げられる.

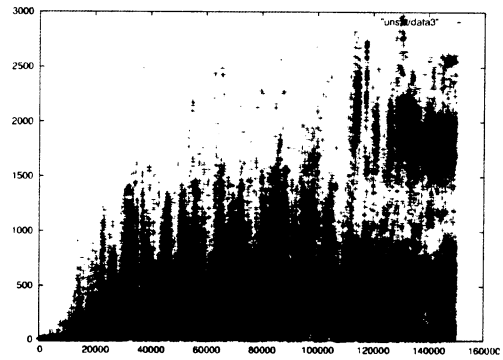

図 3 探索の進行と得られる lemma の長さ

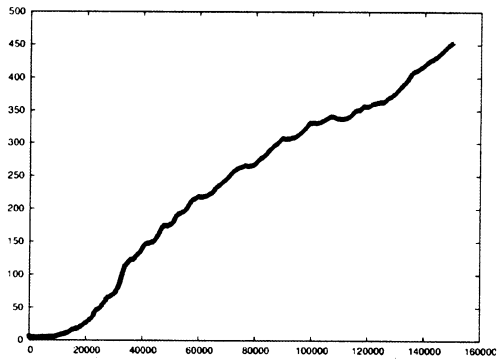

図 4 探索の進行と lemma の平均長

4.3 MPI-zchaff の実装

本研究で実装した並列 SAT ソルバ MPI-zchaff では前述の問題を避ける事と, 全計算ノードが常に探索している事を目標として以下のような設計をした.

(1) 各計算ノードはそれぞれ別の探索域を並列に探索するようにする.

(2) 探索中に lemma を交換する事によりノード間の探索域の重複をさける.

具体的には MPI-zchaff ではホストとクライアントに別れて探索を行う. lemma の通信及びクライアントの管理部分を MPI で記述した. 各クライアントのソルバコアに zchaff を採用し, それぞれのノードに別の探索域を探索させるために, 通常の変数選択にランダムなパラメタを加えた zchaff を実行する (1 台はデフォルトパラメタのまま). zchaff は与えられたパラメタを使い VSIDS の通常の選択からずらした別の変数に先に真偽値を割り当てる. 参考文献 2) では「ある変数選択をすると 1 日かかる問題が別の変数を先に真偽値を割り振った結果, 数時間で終わる事がある」と述べられている.

各ノードは一定時間経過後にホストと lemma を送受信し, 他のノードの学習効果を探索に取り入れることで, 探索域の重複を避けながら効率的に探索を進める.

各ノードが交換する lemma について前述した lemma のトレードオフの観点から考えると, 全ての lemma の交換は探索プロセス自体の遅れと, ネットワークの遅延による探索の遅れを招いてしまう事が予想されるので, 探索域の削減効果の高い lemma のみを交換するようにしなければならない. 一般的に探索域削減効果の高い lemma は変数の数が少ない lemma であると考えられるので MPI-zchaff では lemma の長さに制限を付けてホストとの送受信を行う.

図 3 は zchaff で充足不可能な問題における, 探索時間と共に得られる lemma の長さの関係であり, 図 4 は探索の進行と共に変化する lemma の平均長を示す. 図の縦軸は変数の個数であり, 横軸は探索で得られた順序である.

lemma の学習は前章で述べたように implication graph における最小グラフカット問題と見なすことができ, 探索が進み学習と割り当てを繰り返すと implication graph が複雑化するために得られる lemma の平均長が長くなっていくものと考えられる. MPI-zchaff では毎回の探索の中で得られた lemma の中で平均より短いものを交換する.

全体として探索域削減効果のあまり無い lemma を他のノードが取り入れてしまうという事が考えられるが, lemma の長さの制限値を MPI-zchaff 全体として管理する方法などの解決案が考えられる.

5. 性能評価

測定環境には Myrinet で接続された Dual Pentium III 1.26GHz 16 ホスト構成のクラスタを用い, 実行環境として SCore 5.6.1 を用いた.

SAT Competiton 2002, 2003 にて実際に使用された問題を使って性能測定をした. 表中の値は各問題あたり 5 回の実行時間を測定した平均値である. 表 1 から表 2 の測定結果の単位は秒であり, Ratio は zchaff を 1 台で実行した時の場合と MPI-zchaff を用いた場合の性能向上比を示す.

見する可能性が高まるためと思われる．これは単体のソルバでも言える事で，walksat などの確率的アルゴリズムのソルバに衝突学習アルゴリズムを付与する事で，ハードウェア検証など，探索域に対称性のある問題において性能向上が見込めるだろう．

5.5 探索時間

現在の実装では zchaff のコアに TIMEOUT をかけ，lemma の交換までの探索時間を決めているが，unit clause rule が大半の時に 探索を一旦打ち切って lemma の送受信をすると，遅くなってしまう事がある．これを分岐回数で探索を区切るように改善すれば平均的に高速にできると考えられる．

6. まとめと今後の課題

本研究では SAT ソルバをクラスタ上で効率的に並列化する手法を提案した．本研究の並列 SAT ソルバは学習効果の高い lemma の交換と，確率的 SAT ソルバのようなランダムな探索域の探索により，解の探索時間を非常に高速化する事ができた．これにより SAT Competition 2003 においてどの単体の SAT ソルバも解けなかった問題の充足性の判定もできた．一方，充足不可能な問題においては本実装では探索が進み探索域が小規模になった時には各ノードの探索域が重複しやすいことと，ランダムな変数選択は充足不可能な事をいち早く示すにはあまり良い変数選択でないために，低い性能向上率であったと考えられる．今後の課題として，

- より効率的な探索域の負荷分散の研究・実装
- zchaff 以外の高速 SAT ソルバの並列化
- 並列 SAT ソルバの応用分野への適用

などを考えている．

参考文献

1) 大橋智昭: SAT ソルバ zchaff の MPI による並列化, 2003 年度卒業論文, 早稲田大学理工学部, 2004.
2) L. Zhang, S. Malik: The Quest for Efficient Boolean Satisfiability Solvers. In *Proc. 8th Int. Conf. on Computer Aided Deduction (CADE 2002)*, July 2002.
3) L. Zhang, C. Madigan, M. Moskewics and S. Malik: Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. In *Proc. Int.Conf. on Computer Aided Design (ICCAD2001)*, pp. 279–285 Nov 2001.
4) SAT Research Group at Princeton University: zChaff Website, 2004. `http://ee.princeton.edu/~chaff/zchaff.html`
5) W. Chrabakh, R. Wolski: GridSAT - A Chaff-based Distributed SAT Solver for the Grid. In *Proc. the ACM/IEEE SC2003 Conference*, Nov 2003.
6) B. Jurkowiak, C. M. Li, G. Utard: //Satz: A Parallel SAT Solver with Dynamic Load Balancing. In *Proc. Workshop on Theory and Applications of Satisfiability Testing (SAT2001)*, pp. 205–211, June 2001.
7) J.Erenrich: WalkSAT-MPI Website. `http://www.jordanerenrich.com/walksat/`
8) SAT Competitions Website. `http://www.satlive.org/SATCompetition/`
9) 上田盛慈, 鵜飼訓史, 井上克巳, 番原睦則, 田村直之, 川村尚生: SAT ソルバの並列実行に関する一考察, 電子情報通信学会技術研究報告, AI2003-8, pp. 41–46, May 2003.
10) C. Sinz, W. Blochinger, and W. Küchlin: PaSAT - parallel SAT-checking with lemma exchange: Implementation and applications. In *Proc. Workshop on Applications of Satisfiability Testing (SAT2001)*, June 2001.
11) W. Gropp, E. Lusk, A. Skjellum: *Using MPI* - 2nd edition, The MIT Press, 1999.
12) Peter S. Pachevo 著, 秋葉博 訳: MPI 並列プログラミング, 培風館, 2001.

表 1 SAT Problem の実行時間 (単位:秒)

Problem	lisa19_1	lisa19_3	lisa19_0	lisa20_1	lisa20_3	sat03-1719	sat03-1723	sat03-1808	sat03-1809
satisfiability	SAT	SAT	SAT	SAT	SAT	SAT	SAT	SAT	SAT
zchaff	66.5	81.8	78.6	445.0	386.0	-	-	-	-
MPI-zchaff	41.9	2.2	11.4	2.8	8.6	84.7	25.6	263.9	174.2
Ratio	1.59	37.1	6.8	158.9	44.8	-	-	-	-

表 2 UNSAT Problem の実行時間 (単位:秒)

Problem	v250-1	v250-2	v300-1	v300-2	v300-3	v300-4	ferry10u	hanoni4u
satisfiability	UNSAT	UNSAT	UNSAT	USAT	UNSAT	UNSAT	UNSAT	UNSAT
zchaff	7.6	8.5	178.2	287.6	99.1	329.8	0.16	29.3
MPI-zchaff	8.4	9.5	64.0	68.7	48.8	83.8	0.16	29.8
Ratio	0.9	0.9	2.7	4.1	2.0	3.9	1.0	1.0

表 3 zchaff と MPI-zchaff 解いた問題総数の比較

	SAT Problem	UNSAT Problem	合計
zchaff	62	52	114
MPI-zchaff	118	58	176

表 4 lemma と性能向上

Satifsfiability	SAT	SAT	SAT	SAT	UNSAT	UNSAT	UNSAT
lemma 送受信なし	1.8	32.8	333.8	-	31.3	50.1	114.9
lemma 交換	2.0	24.8	64.9	391.8	30.4	37.1	84.2

5.1 SAT Problem

表 1 は問題に解がある問題の zchaff と MPI-zchaff の性能比較である．解の探索において MPI-zchaff は zchaff の単体の場合に比べて，ほとんどの問題で大きく性能向上している．また，解を発見したノードはどの問題でもランダムなパラメタを加えて探索をしているノードであった．

5.2 UNSAT Problem

表 2 は充足不可能な問題の zchaff と MPI-zchaff の性能比較である．SAT Problem と比較して全体的に性能向上率が低い結果となった．また充足不可能な事を示したノードはデフォルトのパラメタで探索をしているノードだった．性能が低い原因としては，探索が進むとノード間で探索域が重複してしまう事，ランダムな変数選択では充足不可能な事をいち早く示すために良い変数選択をしてない事，分岐が少なく，unit clause rule が実行時間の大半を占めるような問題では現在の実装では並列効果が出にくいこと，などが考えられる．

5.3 判定問題数の比較

表 3 に MPI-zchaff と単体の zchaff の解いた問題の総数を示す．SAT Problem では zchaff で解けなった問題が多数解けるようになっている．SAT Competition 2003 でどのソルバも単体の計算機では制限時間内に充足性の判定ができなかった問題の解を MPI-zchaff で発見することができた．一方，充足不可能問題はでは元の zchaff と大差のない結果となった．本実装では 1 台を除いたノードがランダムに探索域を決定しているため，今後充足不可能問題での性能向上のためのより効果的な動的負荷分散の実装が必要であろう．

5.4 lemma の効果

lemma による性能向上を評価するために，MPI-zchaff において lemma の交換をしない実装と，通常の lemma を交換するものを比較した．表 4 にその結果を示す．充足不可能な問題においては，最初に解の判定をしたのが全て変数選択が通常のパラメタのままのノードだったため，lemma の送受信なしの場合は単体の zchaff と同じ性能であった．一方，充足可能な問題においては lemma の探索域削減効果は高く，探索に時間がかかる探索域の広い問題の方が大きく性能差が現れた．充足可能な問題における性能向上は，lemma により探索域をある程度削減された探索域に対して，台数分のランダムな探索を行うので，解の発