計算機アーキテクチャ 137−17
ハイパフォーマンス
コンピューティング 80−17
（2000. 3. 3）

PVMによるSAT並列局所探索プログラム

梅本 潤　宮崎 修一　岡部 寿男　岩間 一雄

京都大学 大学院情報学研究科

〒606-8501 京都市左京区吉田本町

Tel 075-753-5388

Fax 075-753-5972

{umemoto,shuichi,okabe,iwama}@kuis.kyoto-u.ac.jp

充足可能性問題(SAT)は基本的な組み合わせ問題である．高速なSATの解法アルゴリズムとして局所探索法があるが，大規模な問題にも対応するため更なる高速化の重要性が高まっている．そこで，本稿では局所探索法をネットワーク並列計算用ソフトウェアシステムであるPVM(Parallel Virtual Machine)で並列化して実行する高速化手法を提案する．局所探索法のTRYという実行単位を各計算機で並列に実行するTRY同時実行により，効率的に高速化できることを示した．さらに，*2nd DIMACS Implementation Challenge*のsatisfiabilityのベンチマークに対して計算機実験を行い，これまで解かれていなかった例題をワークステーション70台を用いて8,500秒程で解くことができた．

A parallel local search program for SAT using PVM

Jun Umemoto　Shuichi Miyazaki　Yasuo Okabe　Kazuo I wama

Graduate School of Informatics, Kyoto University

Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501

Phone +81-75-753-5388

Fax +81-75-753-5972

{umemoto,shuichi,okabe,iwama}@kuis.kyoto-u.ac.jp

The Satisfiability Problem (SAT) is one of the most fundamental combinatorial problems. The local search algorithm, which was developed recently, is probably the most popular SAT algorithm because of its surprising efficiency. The purpose of this paper is to speed up local search using PVM (Parallel Virtual Machine), which is the software system designed for networked parallel computing. We show that the local search algorithm can be efficiantly speeded up by the method 'TRY simultaneous execution'. The method executes TRYs, the unit process of the local search, on several computers in parallel. Furthermore, we show experimental results using benchmarks of *2nd DIMACS Implemaentation Challenge*. In this experiment, we were able to solve an instance, which has not been solved by other programs, in about 8,500 seconds using 70 workstations.

1 はじめに

CNF論理式の充足可能性問題(SAT)は基本的な組合せ問題であり，多くの計算理論分野で登場する重要な問題である．MAXクリーク問題やグラフのカラーリング問題のような組合せ最適化問題はSATに変換することができ[3]，また，論理設計の分野でも，故障検査問題などを

SAT に変換することができる．さらに，講義の時間割作成 [3] や学生が研究室に配属される際の成績・定員による振り分け [13] といった実世界の様々な問題が，適当な変換アルゴリズムで SAT に変換できることが示されている．実際に SAT を解くプログラムにより，このような問題を計算機上で解決できるので，解法アルゴリズム，例題生成アルゴリズム等の研究が盛んに行われている [1, 2, 6, 8].

しかし，SAT は変数数・項数が大きくなると実用的時間で解くのは困難であると考えられているため，近年では，実用的時間でできるだけ多くの項を充足する解を見つける近似解法の研究も行われている [5]. その中で，特に局所探索法 (Local Search) は最適に近い解を高速に見つけるとの報告もあり，注目を集めている [1, 2]. 局所探索法を実装した場合，解ける例題のサイズはアルゴリズムと共に，スピードに大きく依存する．従って，高速な SAT の解法プログラムの開発は重要な課題である．

そこで，本研究では PVM(**P**arallel **V**irtual **M**a-chine)[4] を用いて局所探索法を高速化することを目標とする．PVM はネットワークに接続された異機種 UNIX コンピュータ群を，単一の並列コンピュータとして利用することを可能にするソフトウェアシステムである．局所探索法のプログラムには GSAT[9] を高速化したもの [12] に，Random Walk[8, 9, 10] および Weighting[1, 2] をそれぞれ組み合わせたものを開発した．これを PVM を使い，局所探索法の TRY という実行単位を各計算機で並列に実行する TRY 同時実行により，効率的に高速化できることを示した．

性能評価は，京都大学内でワークステーション最大 70 台の規模で，また，広島大学，九州大学の協力により大学間を結んでの実験も行った．その結果ほぼ使用した計算機の台数に比例して高速化できることを示した．さらに，ワークステーション 70 台を用いて *2nd DIMACS Implementation Challenge*[8] のベンチマークを行い，現在まで他のプログラムにより解かれていなかった例題を約 8,500 秒で解くことに成功する等，高い性能を確認できた．

2 SAT および局所探索法

$x_i (i = 1, 2, \ldots)$ を**変数**といい，これらは真 (1) または偽 (0) の値を取る．変数 x_i およびその否定 $\overline{x_i}$ を**リテラル** といい，リテラルの論理和を**項**と呼ぶ．また，項を論理積で結合して得られた論理式を**和積形 (CNF) 論理式**と呼ぶ．充足可能性問題 (SAT) とは，CNF 式 f が与えられたとき，f のすべての項を 1 にする割当が存在するかどうかを問う決定問題である．

SAT において，全ての変数に対する 0 か 1 の変数割当 C_1 と C_2 の間で 1 変数だけ割当が異なる場合，C_1 と C_2 は互いに**隣接する**という．

局所探索法 [7, 9, 10] は近年開発された SAT アルゴリズムであり，基本的には以下のように動作する．まず，変数割当を最初にランダムに選び，それを変数割当 C とする．次に，C に隣接する変数割当の中で，0 となる項の数が C よりも少なくなる変数割当があれば，その中で最も少なくなる変数割当に移動し，その変数割当を C として，この処理を繰り返す．そして，最終的には 0 となる項の数が 0 の変数割当に到達しようとするものである．また，隣接する変数割当への移動の際，変数 x_i の割当を変えることを変数 x_i を**フリップ**するという．

変数割当 C に隣接するすべての変数割当の中で C より良い変数割当がないとき，C を局所解という．通常の局所探索法適用中に局所解に到達してしまうと，変数割当の移動が起こらなくなり，局所探索法は停止してしまう．局所解に到達した場合に局所解から脱出する手法がいくつか知られている．

GSAT [9]
: 充足解を得るために局所解に到達しても，隣接するすべての変数割当の中で最も良いものを次のステップの変数割当とする．したがって，良さが同等の変数割当，あるいは，より悪い変数割当に移動することも許される．これにより，何度か到達する局所解をくぐり抜けて充足解に到達しようとするものである．

Random Walk [8, 9, 10]
: 2CNF 論理式に対する有効な手法に以下

のようなものがある.

1. ランダムな割当を選ぶ
2. ランダムに充足されていない項を選ぶ
3. その項に出現する変数を一つ選びフリップする.
4. 2, 3 を全ての項が充足されるまで繰り返す.

この手法により, 2SAT を確率的に $O(n^2)$ で解くことができる (n:変数数)[9]. しかし, この手法は一般の CNF 式には有効ではない. random walk はこの手法と GSAT を組み合わせたもので, フリップする変数を決定する際に, 確率 p で上記の手法をとり, 確率 $1-p$ で GSAT の手法をとる. p の値はランダム例題では 0.5 から 0.6 の間が良いとされている [9].

Weighting [1, 2]
穴 (局所解) に落ち込んだら, 穴を埋めることによって穴から脱出しようというものである. すなわち, 局所解での 0 である項に重みを与える (その項をもう一度 CNF 式 f の中に繰り返す). そうすれば現在の変数割当 (局所解) での 0 である項の数 (0 である項の重みの総数) が増加して, 現在の変数割当から, 別の隣接する変数割当に移動できるようになる.

3 SATプログラムの実装

局所探索法による SAT プログラムを MAXFLIPS, MAXTRIES という 2 つのパラメータを用いて実装した. ランダムに初期割当を選び, フリップを MAXFLIPS 回行うことを 1TRY という. 局所探索法は TRY を MAXTRIES 回行う. この過程で充足解が見つかれは, その時点で実行は終了となる.

GSAT の高速実装は我々の研究グループで研究されている [12]. フリップの高速化には, 次のような GAIN とバケットを使った手法が知られる. ある変数をフリップすれば, どれだけ 0 である項が減るか (変数の GAIN という) を記憶しておき, GAIN の値が最大である変数をフリップする変数とする. GAIN が最大である変数を探索するのは, 変数数のオーダーのコストがかかるので変数を GAIN>0, GAIN=0, GAIN<0 の 3 つのバケットにグループ分けし, 1 つのバケットから GAIN が最大の変数を探索する. [12] ではさらにバケットを GAIN>0, GAIN=0 の 2 つに効率化し, 各変数がバケットのどの位置にあるかを記憶することにより GAIN の更新を高速化した.

本研究では [12] の GSAT をベースに, 次の 2 つの SAT プログラムを作成した.

RWSAT GSAT[12] をベースに Random Walk を行うプログラム

WSAT GSAT[12] をベースに Weighting を行うプログラム

4 TRY同時実行による並列化

局所探索法の各 TRY は互いに独立である. 従って, 複数の計算機で局所探索を実行し, それを PVM を用いて統合すれば, 複数の TRY を同時に実行する並列局所探索を実現できる. 局所探索法ではランダムに初期割当を選ぶために乱数の扱いが重要であるが, PVM により各プログラムに割り当てられるタスク ID と時刻関数を乱数の種に用いることにより, 複数のプログラムでの TRY 同時実行と, 同じ回数 TRY を逐次実行した場合の成功率を同じとすることができる. 本稿ではこれを TRY 同時実行による並列局所探索法と呼ぶ. 全体を統合するため PVM で通信を行うのだが, 各 TRY で解けたかどうかを通信するだけなので大きなコストはかからず, ほぼ計算機数に比例して高速化できると考えられる.

具体的な実装は, マスターとスレーブの 2 つのプログラムからなる. スレーブプログラムは局所探索プログラムそのものであり, マスタープログラムがそれをまとめる役割を果たす. 各 SAT プログラムは 1 TRY 毎に成功したかどうかをマスターに通信する. マスターは通信を受け, 成功なら各スレーブを終了させ実行終了となり, 失敗の通信なら TRY の回数を数えて設定の MAXTRIES 回を越えた時点で実行終了し, 解は得られなかったということになる.

通信のコストを考えなければ, 並列度(計算機台数, 性能は同じとする)が n とすると, 並列局所探索法は通常の局所探索法に比べて同じ時間で n 倍の TRY を実行できる. ただし, 1 台の計算機ではマスターとスレーブが同時に実行されることになるが, マスターの必要とする処理は小さく大きな影響はない. 同じ回数 TRY を実行すれば成功率は同じなので, 並列局所探索法は通常の局所探索法に比べ, 同じ成功率を得ようとする場合には並列度 n に比例して高速化されることになる.

5 性能評価

5.1 1秒あたりのフリップ回数に関する実験

前節で述べたように, 通信のコストを除けば TRY 同時実行による並列局所探索法は通常の局所探索法に対し, 並列度に比例して高速化できると考えられる. そこで計算機実験により通信のコストも含めて実際にどの程度高速化されるかを検証する. この実験では局所探索法の手法として GSAT を使用した. フリップ回数に注目し, 成功率については考えていない.

また, LAN 内での通信とより大きな範囲での通信ではコストがかなり違うと考えられるので, LAN 内での通信のみの場合と LAN 外との通信も含む場合で 1 秒あたりのフリップ回数を比較する.

実験では 100 変数と 300 変数のランダム例題 100 個ずつに対して, MAXFLIPS =(変数数) × 10, MAXTRIES = (ホスト数) × 10 の並列局所探索法を実行した.

図 1 は並列度に対する 1 秒あたりの平均フリップ回数, 図 2 は LAN 内での通信のみの場合と LAN 外との通信も含む場合で 1 秒あたりのフリップ回数の比較である.

LAN 内の実験では本研究グループのマシンのみを使用した. LAN 外では, GSAT を実行するマシンは LAN 内の場合と同じで, マスタープログラムだけを広島大で実行した. つまり, 通信は全て京都大学と広島大学の間で行われている.

図 1 の実験では, 100 変数, 300 変数ともにほぼ並列度に比例して高速化されている. ただし, 並列度が 64 まで上がると速度向上率が低下しているので, 並列度が上がって通信の頻度が増えた結果と思われる.

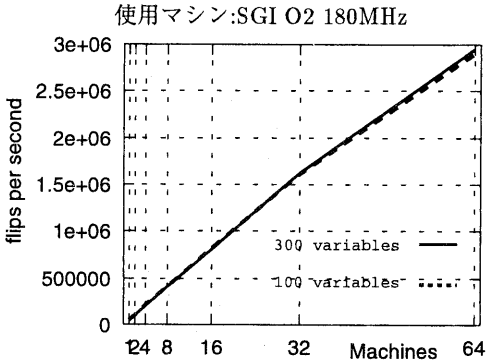

図 1: 並列度に対する 1 秒あたりの平均フリップ回数

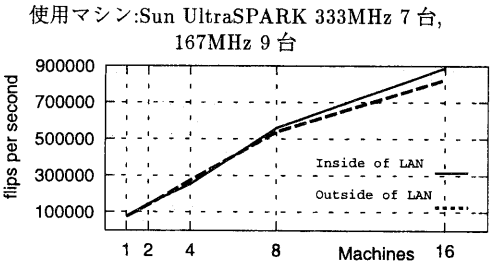

図 2: LAN 内と LAN 外の通信の比較(100 変数)

図 2 の実験では, LAN 内を通る通信も LAN 外(京都大学, 広島大学間)を通る通信でもほぼ同様の高速化が得られている. この実験で使用マシン 16 台での速度の低下は, マシンの性能にばらつきがあるためである.

2 つの実験から, TRY 同時実行の並列化では通信の影響は大きくなく, ほぼ予想通りに高速化できると考えられる. 特に, 1 つ 1 つの通信のサイズが小さいため遅い通信路でも大きな影響はみられない. 通信の頻度が高い場合は多少影響を受けるが, SAT の問題のサイズが大きいほど通信の頻度は下がるため深刻な問題ではないと言える.

5.2 DIMACS ベンチマーク

次に, 我々の TRY 同時実行プログラムの性能を客観的に評価するため, *2nd DIMACS Implementation Challenge*[8] の SAT ベンチマークに対する実験を行った. 表 1 が解が得られるまでの実行時間の結果であり, [8] に紹介された 7 つのプログラム及び, [12] の我々の研究グループで開発されたプログラムの結果と比較している. RWSAT 及び WSAT は表 2 のような環境で実行し, 表 1 には 70 台での並列実行の結果と, 1 台で実行させた結果をあわせて示す. 空欄は実行していないことを表し, N.A.(No Answer) は解けなかったことを表す. なお, 我々の実験結果に関しては 6 時間までで解けないものは N.A. とした.

我々の実験結果を見ると, もともと高速に解けるものも多いが, 随所に並列化の効果が現れ, 実行時間が短縮されている. 特に我々が注目するのは, ランダムに生成された 3CNF 論理式である f 例題の結果である. RWSAT はランダム例題に高い能力を示している. f1600 例題は現在まで他のプログラムでは解かれておらず本稿の 70 台並列 RWSAT によって初めて解かれた例題である.

また, GSAT をベクトル計算機 VPP800/63[11] でベクトル化し, PVM を用いて 40 個の CPU で並列実行するプログラムである [12] との比較を考えると, 我々のプログラムが多くの例題において優位に立っている. GSAT とのアルゴリズムの能力の差によるところも大きいが, CPU の台数分の高速化を得るという点では, 我々が提案する TRY 同時実行の局所探索では, プロセッサ間の通信がほとんど発生しないため, 表 2 のようなワークステーションを FastEthernet で接続したクラスタ環境においても, VPP800 のような高速のプロセッサ間結合網を持つスーパーコンピュータと比べて遜色のない性能が得られると言える.

6 おわりに

PVM を使用した TRY 同時実行による局所探索法の並列化手法を提案した. この並列化手法は非常に単純だが, 通進路の性能に影響を受けにくく, 効果的に高速化が可能であることが確認できた. 本稿で提案した PVM による並列化は 3 節で述べた TRY という実行単位をとるアルゴリズムであれば, どのようなアルゴリズムにも適用できるので, 優れたアルゴリズムにこの並列化手法を組み合わせることで簡単な労力で高速なプログラムを開発することができる.

参考文献

[1] B. Cha and K. Iwama, "Performance Test of Local Search Algorithms Using New Types of Random CNF Formulas", *Proc. IJCAI95*, 1995.

[2] B. Cha and K. Iwama, "Adding new clauses for faster local search", *Proc. AAAI96*, 1996.

[3] B. Cha, K. Iwama, Y. Kambayashi and S. Miyazaki, "Local search algorithms for Partial MAXSAT", *Proc. AAAI97*, 1997.

[4] A. Gaist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sanderam, "PVM:Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing", The MIT Press, 1994.

[5] M. X. Goemans and D. P. Williamson, ".878–approximation algorithms for MAX CUT and MAX 2SAT", *Proc. STOC*, pp. 422–431, 1994.

[6] K. Iwama, "CNF satisfiability test by counting and polynomial average time", *SIAM J.Comput.*, pp. 385-391, 1989.

[7] E. Koutsoupias and C. H. Papadimitriou, "On the greedy algorithm for satisfiability", *IPL43*, 53-55, 1992.

[8] D. S. Johnson and M. A. Trick, Eds, "Cliques, Coloring, and Satisfiability, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science volume 26*", American Mathematical Society, 1996.

[9] B. Selman, H. Levesque, D. Mitchell, "A New Method for Solving Hard Satisfiability Problems", *Proc. AAAI92*, 1992.

[10] B. Selman, H. Kautz, "Local Search Strategies for Satisfiability Testing", *Second DIMACS Implementation Challenge Workshop*, 1993.

[11] 京都大学大型計算機センター, "ベクトル並列計算機 VPP800/63", `http://www.kudpc.kyoto-u.ac.jp/Supercomputer/`

[12] 河合大輔, 宮崎修一, 岡部寿男, 岩間一雄, "SAT に対する局所探索法のベクトル化", 電子情報通信学会技術研究報告 COMP99-69 ~ 78, COMP99-75 pp.49-56, 2000.

[13] 盛田保文, 宮崎修一, 岩間一雄, "部分 MAXSAT を利用した大学情報処理の自動化", 情報処理学会研究報告, Vol.98 No.58, 98-DBS-116(2) pp.335-342, 1998.

表 1: DIMACS ベンチマークに対する結果

例題名	変数数	項数	実行時間 (秒)								
			RWSAT	WSAT	他の SAT プログラム [8]						
			70 台/1 台	70 台/1 台	[12]	1	2	3	4	5	6	7
aim-100-2_0-yes1-1	100	200	14/38	0/0	1	0	17	135	2	398	N.A.	1
aim-100-2_0-yes1-2	100	200	17/73	0/0	2	0	29	2	5	239	13,929	1
aim-100-2_0-yes1-3	100	200	15/57	0/0	1	0	13	17	1	63	22,500	1
aim-100-2_0-yes1-4	100	200	47/468	0/0	2	0	15	0	0	1,456	N.A.	0
f400.cnf	400	1,700	1/1	1/0	3	2,844	34		5,727	210,741	60	10,870
f800.cnf	800	3,400	4/163	15/1038	182		1,326				27,000	
f1600.cnf	1,600	6,800	8,564/N.A.	N.A.	N.A.						N.A.	
g125.17.cnf	2,125	66,272	16,562/N.A.	13,156/N.A.	261		103,310			N.A.	453,780	N.A.
g125.18.cnf	2,250	70,163	15/178	8/27	17		126			N.A.	480	N.A.
g250.15.cnf	3,750	233,965	9/7	6/2	17		72				120	
g250.29.cnf	7,250	454,622	N.A.	N.A.	2,751						398,820	
ii32b3.cnf	348	5,734	1/0	1/0	15	2	4	4	1	1	5,400	17
ii32c3.cnf	279	3,272	1/0	1/0	8	1	3	0	5	1	12,180	
ii32d3.cnf	824	19,478	2/1	1/1	38	973	19	10	3	20	1,200	N.A.
ii32e3.cnf	330	5,020	1/0	1/0	11	1	3	4	1	3	3,900	3
par16-2-c.cnf	349	1,392	1,432/9,083	11/147	183	23		329	48	1,464	N.A.	145
par16-4-c.cnf	324	1,292	226/5,904	3/66	554	6		210	116	1,950	N.A.	145
par32-2-c.cnf	1,303	5,206	N.A.	N.A.	N.A.						N.A.	N.A.
par32-4-c.cnf	1,333	5,326	N.A.	N.A.	N.A.						N.A.	N.A.
par8-2-c.cnf	68	270	1/0	0/0	1	0	2	0	0	0	1	0
par8-4-c.cnf	67	266	1/0	0/0	1	0	4	0	0	0	12	0
ssa7552-038.cnf	1,501	3,575	1/2	1/0	N.A.	1	152	0	3	2	N.A.	2
ssa7552-158.cnf	1,363	3,034	1/1	0/0	N.A.	1	83	43	1	1	N.A.	1
ssa7552-159.cnf	1,363	3,032	1/1	0/0	N.A.	1	82	6	1	1	N.A.	1
ssa7552-160.cnf	1,391	3,126	1/0	0/0	N.A.	1	86	6	1	23	175,500	1

表 2: RWSAT, WSAT の実行環境

計算機	クロック数 (MHz)	台数	ネットワーク
SGI O2	180	70	FastEthernet

表 3: 表 1 中の番号と SAT プログラムの対応 [8]

プログラム No.	著者	プログラム名	Ratio*
1	O.Dubois, P.Andre, Y.Boufkhad, J.Carlier	C-SAT(backtracking)	1.64
2	Steven Hampson, Dennis Kibler	Cyclic,Opportunistic Hill-Climing	3.63
3	Brigitte Jaumard, Mihnea Stan, Jacques Desrosiers	Tabu Search	5.49
4	Daniele Pretolani	H2R,BRR(pruning)	9.10
5	M.G.C. Resende, T.A. Feo	Greedy Randomized Adaptive Search Procedure (GRASP)	0.93
6	William M. Spears	SASAT(Simulated Annealing)	2.39
7	Allen Van Gelder, Yumi K. Tsuji	2cl(Combination of branching and limited resolution)	1.51

* Ratio: 我々の使用した計算機 (1 台) に対するプログラム実行計算機の計算時間比