電気学会論文誌 D（産業応用部門誌）
IEEJ Transactions on Industry Applications
Vol.139 No.2 pp.119–126　DOI: 10.1541/ieejias.139.119

論　文

並列SATソルバのXeon Phiプロセッサ上での性能評価

非会員　西脇慎太朗*　非会員　藤枝　直輝*a)　正　員　市川　周一*

Performance Evaluation of Parallel SAT Solver on Xeon Phi Processor

Shintaro Nishiwaki*, Non-member, Naoki Fujieda*a), Non-member, Shuichi Ichikawa*, Member

（2018年3月23日受付，2018年9月18日再受付）

The satisfiability (SAT) problem is widely applicable to various industrial problems, such as formal verification of industrial systems. This study presents the performance evaluation of a parallel SAT solver (Glucose Syrup) on Intel Xeon Phi many-core processors. After limiting the number of learnt clauses to be shared, the number of solved problems in the SAT Competition 2016 workload increased from 201 to 236, yet it was smaller than that of a Xeon processor.

キーワード：メニーコアプロセッサ，計算困難問題，並列化
Keywords: many-core processor, computationally hard problems, parallelization

1. はじめに

充足可能性問題（Satisfiability Problem；SAT）は，ある命題論理式が与えられたときに，その値を真とする変数の割当てが存在するか判定する問題であり，NP完全問題の代表例の1つである。SATを解くアプリケーションをSATソルバとよぶが，2002年以降はSAT Competition，SAT Raceなどの国際競技会が毎年開催されており，こうした競技会を通じたSATソルバの改良により，大規模な問題でも現実的な時間で解を求められるケースが増加している。そのため，SATはソフトウェア[1]，ハードウェア[2][3]を問わず，システムの検証に広く応用されている。こうした競技会の課題データセットにも，例えば鉄道の連動装置の安全性の形式的証明[4]といった産業応用に関する問題が含まれており，SATソルバの高速化は，産業システムの設計・検証に大きな役割を果たしている。

一方，市販のマイクロプロセッサにおいては，単一プロセッサコアでの性能向上が困難となったことから，2005年頃を境に複数・多数のプロセッサコアをワンチップに集積するマルチコア化・メニーコア化が進んでいる[5]。単一コアあたりの性能を抑制するかわりに，複数コアで並列処理を行うことで，システム全体の性能向上が期待できる。あるいは，並列処理をコプロセッサ（補助演算装置）に分担させることもしばしば行われる。例えば，画像処理や機械学習の分野ではGPU（Graphic Processing Unit）が広く用いられている。コプロセッサはCPUと異なるアーキテクチャを持つため，その特性により得意・不得意とする計算が異なる。

本稿では，Intel社のメニーコアプロセッサまたはコプロセッサであるXeon Phiを用いて，並列SATソルバの1つであるGlucose Syrup[6]の性能評価を行う。Xeon Phiは，High-performance Linpackなどの高性能計算に代表される，高並列・低通信のアプリケーションについては性能が発揮されることがわかってきているが，その他の計算についてはまだわかっていないことが多い。また，GPUを用いたSATソルバ[7]や一般のクラスタシステム向けの大規模並列SATソルバ[8]は存在するものの，Xeon Phiのようなメニーコアを対象とした検討はまだ行われていない。そのため，マルチコアCPU向けに並列化されたGlucose SyrupをXeon Phiでも動作するように修正し，実行スレッド数を変化させたときの実行時間を測定し，マルチコアCPUであるXeon上での実行時間との比較を行う。比較により明らかになったXeon Phiの並列SATソルバにおける特性をもとに，高速化の1手法を実装し，評価する。これらによって，将来の高速化のための更なる知見を得ることを目的とする。なお，本論文は2018年3月の電気学会次世代産業システム研究会における発表原稿[9]をもとに，主に高速化手法について

a) Correspondence to: Naoki Fujieda. E-mail: fujieda@ee.tut.ac.jp
* 豊橋技術科学大学 電気・電子情報工学系
〒441-8580　愛知県豊橋市天伯町雲雀ヶ丘1-1
Department of Electrical and Electronic Information Engineering, Toyohashi University of Technology
1-1, Hibarigaoka, Tempaku-cho, Toyohashi, Aichi 441-8580, Japan

加筆したものである。

2. 充足可能性問題（SAT）

〈2･1〉 SAT と SAT ソルバの概要　SAT とは，連言標準形（Conjunctive Normal Form；CNF）で与えられる命題論理式を入力とし，式に含まれる命題変数に 1（真）または 0（偽）を割当てることにより，式全体の値を 1 にすることができるならば充足可能（SAT）を，そうでないならば充足不可能（UNSAT）を出力する問題である。命題変数またはその否定をリテラル（literal）といい，1 つ以上のリテラルの選言を節（clause）という。CNF 式は 1 つ以上の節の連言である。例えば，CNF 式 $(a \vee b \vee c) \wedge (\neg b \vee c) \wedge (\neg a \vee \neg c)$ において，$a = 1$，$b = 0$，$c = 0$ を割当てれば，それぞれの節に含まれる a，$\neg b$，$\neg c$ により節の値を 1 にすることができる。したがって，この式は充足可能（SAT）である。このように，ある式が SAT であることは，その証拠となる割当てを与えることにより容易に検証できる。一方で，ある式が UNSAT であると判断するには，原理的には全ての変数の組合せにおいて式の値が 0 であることを確かめなければならない。そのため SAT は NP 完全問題とよばれる問題の 1 つであり，その計算量は変数の数を n とおけば $O(2^n)$ である。論理合成やスケジューリング問題など，様々な問題が SAT に帰着できることが知られており，SAT の応用範囲は広い。例えば，2 つの論理関数 $f(\mathbf{x})$ と $g(\mathbf{x})$ が同一であることを示すには，$f(\mathbf{x})$ xor $g(\mathbf{x})$ が UNSAT であることを示せばよい。そのため SAT は盛んに研究が行われており，SAT ソルバの改良もそうした研究の方向の 1 つである。

SAT ソルバにおいて現在主流なのは，DPLL（Davis–Putnam–Logemann–Loveland）アルゴリズム [10] に衝突からの節学習（Conflict-driven Clause Learning；CDCL）を追加した，CDCL ソルバ [11][12] である。2016 年の SAT Competition の Main Track において上位となった SAT ソルバはいずれも CDCL ソルバであり，現代の標準的なソルバであるといえる。以下〈2･2〉〜〈2･4〉節では，DPLL アルゴリズムと CDCL，そして本研究のベースとなるソルバである Glucose Syrup [6] について説明する。

〈2･2〉 DPLL アルゴリズム　DPLL アルゴリズム [10] は 1962 年に Davis らにより発表されたアルゴリズムである。DPLL では以下の要領で SAT を解く。

（1） 単位節（1 つのリテラルからなる節）を探す。
　（a） 単位節があればそれを充足する。すなわち，単位節が a であれば $a = 1$ を，単位節が $\neg a$ であれば $a = 0$ を割当てる。
　（b） 単位節がなければ，適当な変数を選択して，その変数に 1 または 0 を割当てる。この操作を選択（decision）という。
（2） 新たな割当てをもとに，各節に対して単位節伝搬（unit propagation）とよばれる以下の手続きを行う。
　（a） もし節が充足されたリテラルをもっているならば，その節自体を消去する。これは吸収則 $a \wedge (a \vee b) = a$ による。
　（b） もし節が充足されたリテラルの否定をもっているならば，そのリテラルの否定を節から消去する。これは分配則および補元則から $a \wedge (\neg a \vee b) = a \wedge b$ であることによる。
（3） 単位節伝搬の結果をもとに，以下のいずれかの処理を行う。
　（a） もし新たな単位節があれば，それをまた充足する。これを含意（implication）とよぶ。
　（b） もし空の節（すべてのリテラルが削除された節）があれば，それは現在の割当てにおいてその節，ひいては式全体が 0 になったことを意味する。これを衝突（confliction）とよぶ。衝突が発生した場合，別の変数選択が可能になるところまで直近の選択を取り消すバックトラックを行う。
　（c） もし新たな単位節も空の節もなければ，まだ割当てられていない変数に対して選択の操作を行う。
（4） 全ての節が消去される（この場合 SAT である）か，全ての割当てを試し終わる（この場合 UNSAT である）まで，（2），（3）を繰り返す。

Fig. 1 に，DPLL アルゴリズムで SAT を解く様子を示す。決定レベル（decision level）は，選択の様子を二分決定木とみなしたときのノードの根からの距離 −1 に対応し，同じタイミングで割当てられた変数をグループ化するために用いられる。まず，単位節がないため適当な変数（ここでは a）を選択して，0 を割当てる（図中の (1)）。これにより $\neg a$ をもつ節は節自体が消去され，a をもつ節は a が消去される。これにより新たな単位節 b が見つかるので，b に 1 を割当てる（図中の (2)）。$\neg b$ は節から消去される。今度は新たな単位節も空の節も見つからなかったので，c を選択し 0 を割当てる（図中の (3)）。c が節から消去されたことにより，$e \wedge \neg e$ が得られ，e に 0 を割当てても 1 を割当てても，どちらかが空の節になってしまう。これが衝突であ

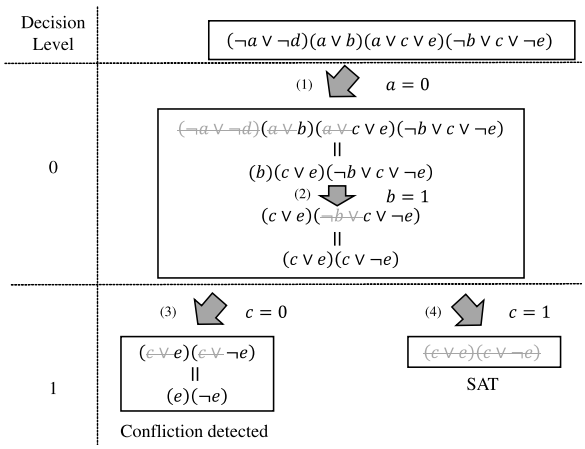


Fig. 1.　An example of the DPLL algorithm.

る。$c = 0$ の割当てを取り消し，かわりに $c = 1$ を割当てる（図中の (4)）。今度はいずれの節も節自体が消去されるので，これにより全ての節が消去された。したがってこの式は SAT であることがわかる。

〈2･3〉 CDCL ソルバ　CDCL とは，DPLL アルゴリズムにおいて衝突が発生した際に，その原因を解析し，原因となる割当てを防ぐための新たな節（学習節）を式に追加することである。Fig. 1 の例では，a と c をいずれも 0 に割当てたことにより衝突が発生しており，a と c を同時に 0 にすると必ず衝突が発生することがわかる。そのため，この場合は $(a \vee c)$ を追加することで，a か c のいずれかに 0 を割当てた段階で，もう一方には必ず 1 が含意されることになり，その後の衝突を回避できる。SAT である問題は探索空間内の 1 つの解を見つければ終了できるが，UNSAT である問題は可能な全ての変数割当てを試さなければならないため，学習節を使用することで探索空間を大きく削減することができる。しかしながら，学習節が探索空間の削減にどれだけ寄与するかは学習節ごとに異なる。計算やメモリ使用量を浪費せずに探索空間を削減するには，削減に効果がある学習節だけを選び，利用することが重要である。そのため CDCL ソルバでは，学習節の数がある程度大きくなった場合に学習節の選別を行い，探索空間の削減効果が見込めない学習節を式から削除する。

〈2･4〉 Glucose Syrup　Glucose Syrup[6] は SAT の競技会において上位の成績を残している Glucose[13] の並列版の SAT ソルバであり，CDCL ソルバの 1 つである Minisat[14] をベースとしている。Glucose のポイントはリテラルブロック距離（Literal Block Distance；LBD）という学習節の指標を導入したことにある。ここでは，同じ決定レベルで割当てられたリテラルの組をブロックとよび，学習節のリテラルがいくつのブロックに分割されるかによって LBD を定義している。LBD が少ない学習節ほど探索空間の削減に効果があるとみなし，こうした学習節を極力残すようにしている。〈2･3〉の例では学習節は $(a \vee c)$ であり，相異なる決定レベルで割当てられていることから，この学習節の LBD は 2 となる。

Glucose Syrup は POSIX スレッドを用いて Glucose を並列化しており，各スレッドが探索空間内の異なる部分を探索しながら，得られた学習節を適宜スレッド間で共有することで，高速化を実現している。Glucose Syrup では，得られた学習節を共有するかどうかの判定にも LBD が用いられる。その閾値は，直近の学習節の選別における選別前の学習節の LBD の中央値である。

3. Glucose Syrup の Xeon Phi 向け実装

〈3･1〉 Xeon Phi の実行モード　Xeon Phi は Intel 社のメニーコアプロセッサまたはコプロセッサである。57〜72 個のコアを集積し，1 コアあたり 4 スレッドの同時マルチスレッディングが可能であるため，200 スレッドを超える並列処理をワンチップで行える特徴がある。ただし，個々のコアの性能は通常の CPU のコアよりも低い。第 1 世代の Xeon Phi はコプロセッサであり，PCI Express を利用してホストコンピュータに接続される一方，第 2 世代の Xeon Phi には通常の CPU と同様にマザーボード上に搭載できるプロセッサバージョンも存在する。

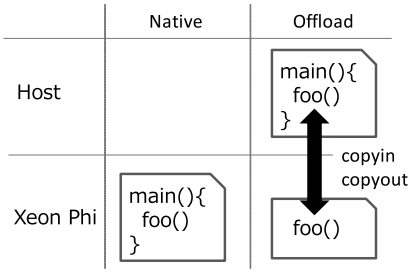


Fig. 2.　Two execution models of Xeon Phi.

第 1 世代の Xeon Phi の実行方法には，ネイティブ（native）実行とオフロード（offload）実行とがある。これらの実行モデルの違いを Fig. 2 に示す。ネイティブ実行はプログラム全体を Xeon Phi に転送して実行するモードであり，ホスト PC はプログラムやデータの転送のみを行う。計算途中のデータを通信する必要がないことが利点である一方，逐次処理を Xeon Phi 上の低速な単一コアで実行しなければならない欠点がある。また，プログラムがファイルの入出力を含む場合，ホスト PC と Xeon Phi 上の RAM ディスクとの間でファイルを転送するか，NFS（Network File System）でホスト PC のファイルシステムを Xeon Phi と共有しておく必要がある。オフロード実行は，まずホスト CPU でプログラムを実行し，Xeon Phi で実行したい関数のみを適宜転送して Xeon Phi 上で実行するモードである。オフロード実行では，ネイティブ実行とは逆に，逐次処理をホスト CPU で高速に実行できるが，計算途中のデータを送受信する時間がかかる。高速な実行のためには，こうした特性の違いを踏まえた上で，アプリケーションに応じて 2 つの実行モードを使い分ける必要がある。

いずれの実行モードでも，Xeon Phi 上で実行する部分に関しては POSIX スレッドなどの並列処理の API がそのまま利用できる。したがって，Glucose Syrup も Xeon Phi 向けに書き換えることができる。以下，本稿では Glucose Syrup を Xeon Phi 上で実行し，性能の測定を行う。

〈3･2〉 オフロード実行の適用　第 1 世代の Xeon Phi で Glucose Syrup を実行するにあたり，実行モデルにはオフロード実行を採用する。というのも，Glucose Syrup では問題データをファイルから読み出すが，4 章で述べる本研究で用いるシステムにおいては，Xeon Phi とホストとの間でファイルシステムの共有を行っていないためである。このような環境でネイティブ実行を行う場合，Xeon Phi の RAM ディスクを圧迫しないよう，あらかじめ問題ファイルをホストから転送しておき，プログラムを実行してから，ホストに実行結果を転送し，問題ファイルを Xeon Phi から削除するといった手続きが必要となる。これに対し，オ

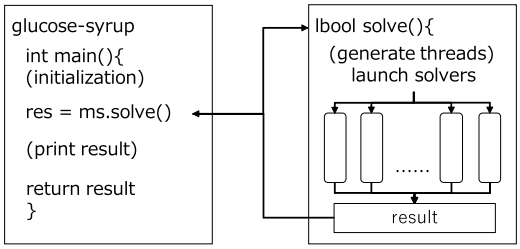


Fig. 3. Basic organization of Glucose Syrup.

フロード実行では手続きが 1 つのプログラムで完結するため，より簡便に利用可能であると考えた。

Fig. 3 に Glucose Syrup の基本構造を示す。初期化（initialization）部分では，問題のファイルからの読み出しやオプションの解釈を行い，ソルバークラスのインスタンスを作成する。次に，作成したインスタンスの `solve()` メソッドを呼び出す。メソッド内で POSIX スレッドによりスレッドが生成され，並列に問題を解き始める。解が得られるか，オプションにより設定された制限時間を超過した場合，結果を出力して終了する。

オフロード実行を適用するにあたっての問題点は 2 つある。第 1 に，オフロード対象の関数は戻り値のない関数である必要がある。これは，戻り値のないラッパー関数を作成することによって解決した。第 2 に，オフロード実行では Xeon Phi にクラスを転送することができない。これには問題データやオプションを Xeon Phi に転送できるデータ型に格納し，ソルバークラスのインスタンスは転送後のラッパー関数で作成することで解決した。

4. 評　価

〈4·1〉 方　法　第 1 世代 Xeon Phi および Xeon における測定には豊橋技術科学大学の広域連携教育研究用クラスタシステムを，第 2 世代 Xeon Phi における測定には京都大学のオープンスーパーコンピュータシステム A を使用した。各システムの実行環境を Table 1 および Table 2 に示す。Table 1 の第 1 世代 Xeon Phi では，ソースに〈3·2〉節で述べた修正を施し，オフロード実行する。Table 2 の第 2 世代 Xeon Phi はプロセッサバージョンであり，修正なしの Glucose Syrup がそのまま動作する。

評価に使用する問題は，SAT Competition 2016 で使用された問題のうち，メモリ使用量と実行時間の観点をもとに，解が UNSAT である問題（UNSAT インスタンス）と SAT であるもの（SAT インスタンス）とを 3 問ずつ選択した。Table 3 に選択した問題のファイル名とファイルサイズを示す。これらは以後 unsat1～3，sat1～3 と略記する。

Xeon では 20 スレッドのみ，Xeon Phi では 40，60，80，100 スレッドの 4 通りの実行スレッド数で評価した。制限時間は 1000 秒とした。Glucose Syrup はスレッドのスケジューリングの影響で実行時間に非決定性があるため，いずれの場合でも 20 回実行し，制限時間内に解けた回数と，実行時間の平均とを評価した。ただし，実行時間の平均を求めるにあたっては，SAT Competition 2017 の規定を参考に，制限時間内に解けなかった場合は実行時間を制限時間の 2 倍（2000 秒）とみなした。

Table 1. Execution environment (TUT).

CPU	**Xeon E5-2680 v2** (2.8 GHz, 10 cores) x2
Memory	64 GB
OS	Red Hat Enterprise Linux 6.2
Compiler	Intel C++ compiler 16.0.0
Coprocessor	**Xeon Phi 5110P** (1.053 GHz, 60 cores), 8 GB of memory

Table 2. Execution environment (Kyoto Univ.).

CPU	**Xeon Phi 7250** (1.4 GHz, 68 cores)
Memory	16 GB + 96 GB
OS	SUSE Linux Enterprise Server 12
Compiler	Intel C++ compiler 16.0.3.210

Table 3. Problems used for evaluation.

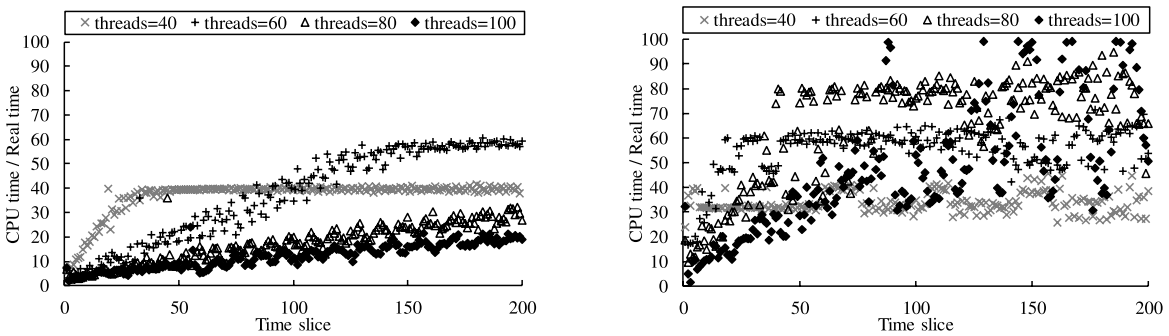
Abbrev.	Problem (File Name)	Size [KB]
unsat1	ecgrid6x115_shuffled.cnf	89
unsat2	fixedbandwidth-eq-40_shuffled.cnf	8
unsat3	homer17.cnf	20
sat1	modgen-n200-m90860q08c40-2087.cnf	154
sat2	modgen-n200-m90860q08c40-6967.cnf	154
sat3	Schur_160_5_d37.cnf	439

〈4·2〉 結　果　Xeon Phi および Xeon における評価結果を Table 4 にまとめる。Threads の列は Xeon Phi における実行スレッド数である（Xeon は全て 20 スレッド）。Solved の列の分子は問題が解けた回数，Time の列は平均実行時間をそれぞれ示す。Xeon Phi ではスレッド数にかかわらず UNSAT インスタンスは 1 度も解けなかったため，これらは 1 行にまとめている。今回選定した 6 つの問題全てで，Xeon Phi は Xeon よりも解けた回数・平均実行時間ともに劣る結果となった。特に，スレッド数を増加させても性能はほとんど変化せず，第 1 世代 Xeon Phi ではむしろ性能が落ちる傾向にあることがわかった。SAT インスタンスの 3 つの問題では，第 2 世代 Xeon Phi は第 1 世代よりも全体的に性能の改善がみられた。

性能低下の原因として，学習節の共有にかかる同期回数の増加が考えられる。各スレッドは自身で得た学習節のうち重要とみられるものを他スレッドに送信しており，また時々他のスレッドから送信された学習節を取り込んでいる。これらの送受信処理には一種のキューが用いられており，各スレッドはキューに学習節を 1 つ投入するたびに，またキューから学習節を 1 つ取り込むたびに，このキューに対するロックを取得している。したがって，スレッドが N 個あるとき，1 つの学習節につきロックは N 回（1 回の投入と $N-1$ 回の取り込み）発生する。各スレッドが単位時間あたりに共有する学習節の数を一定とするならば，単位時間あたりのロックの発生回数は $O(N^2)$ となる。したがって，スレッド数が増加すると大幅に同期回数が増加し，同期待ち時間が増加することによって，性能が低下している可能

Table 4. Evaluation results on Xeon Phi and Xeon. The number of threads on Xeon was 20.

		1st gen. Phi		2nd gen. Phi		Xeon	
	Threads	Solved	Time [s]	Solved	Time [s]	Solved	Time [s]
unsat1	40, 60, 80, 100	0/20	2000	0/20	2000	20/20	365
unsat2	40, 60, 80, 100	0/20	2000	0/20	2000	20/20	502
unsat3	40, 60, 80, 100	0/20	2000	0/20	2000	20/20	739
sat1	40	1/20	1929	5/20	1645	20/20	340
	60	3/20	1805	1/20	1926		
	80	1/20	1944	5/20	1671		
	100	1/20	1933	5/20	1625		
sat2	40	2/20	1865	3/20	1751	15/20	902
	60	3/20	1797	3/20	1731		
	80	0/20	2000	6/20	1597		
	100	0/20	2000	1/20	1928		
sat3	40	1/20	1931	6/20	1598	13/20	962
	60	3/20	1751	7/20	1437		
	80	1/20	1911	4/20	1699		
	100	0/20	2000	4/20	1729		



(a) unsat1　　(b) sat3

Fig. 4. Time change of CPU time per real time on 2nd generation Xeon Phi.

性がある。

また，第 1 世代 Xeon Phi においてはメモリ不足も性能低下の一因となったと考えられる。80 スレッド以上での実行において，計算途中でメモリ不足に陥り，極端に性能が低下したとみられる挙動が観測された。

〈4･3〉 出力ログの解析　性能低下が同期待ち時間の増加によるものである傍証を得るため，Glucose Syrup の出力ログの解析を試みた。Glucose Syrup はデフォルトでは実時間で 5 秒おきにスレッドの衝突発生回数などを集計・出力している。この出力ログをもとにそれまでの CPU 時間を算出し，その差分を取ることにより，そのタイムスライス内での CPU 時間を求めた。これをタイムスライスの実時間（すなわち 5 秒）で割ることにより，実時間あたりの CPU 時間を算出した。この値はその時動作しているスレッド数の平均を示すものであると解釈できるので，その時間変化をプロットすることにより，実際にどの程度並列処理が行われているかを観察することができる。

Fig. 4 に，第 2 世代 Xeon Phi 上で unsat1 および sat3 の求解を試みた場合の，実時間あたりの CPU 時間の推移を示す。グラフ (a) が unsat1，グラフ (b) が sat3 に対する結果を示す。横軸はタイムスライスであり，制限時間が 1000 秒であることから，実行が打ち切られるまでには 200 のスライスがある。縦軸は実時間あたりの CPU 時間であり，40 スレッド実行時から 100 スレッド実行時までを徐々に濃色になるようプロットしている。なお，以前の発表原稿[9]では 3 回の実行結果の中間値をプロットしていたが，本稿では単一の実行の結果をプロットしている。

unsat1 については，スレッド数が増加すればするほど，実行開始直後からのプロットの傾きがゆるやかになっていることが確認できる。40 および 60 スレッド実行時は，最後のタイムスライス付近において実時間あたりの CPU 時間の値が実行スレッド数に近いものとなっており，すなわちスレッドはほとんど停止していない。一方，80 および 100 スレッド実行時は，最後のタイムスライス付近においても 40 スレッド実行時にすら満たない値を示した。このことから，実行の初期において大量の学習節の共有が行われていることと，その同期待ち時間の増加により性能低下が発生している可能性が高いことがうかがえる。

sat3 における 80，100 スレッド実行時では，実時間辺りの CPU 時間の値はタイムスライスごとに大きくばらついているが，最終的には実行スレッド数に近い値を示した。unsat1 との比較から，共有される学習節の量に問題によって大きな違いがあることが示唆される。しかし，やはり実行開始直後からのプロットの傾きは，スレッド数が大きいほどゆるやかである。

以上のことから，同期待ち時間，そのうち特に実行開始

Table 5. Evaluation of the optimizaton method with sat3 on 2nd generation Xeon Phi.

	Threshold = 2		Threshold = 3		Threshold = 4		Threshold = 5		Original	
Threads	Solved	Time [s]	Solved	Time [s]	Solved	Time [s]	Solved	Time [s]	Solved	Time [s]
40	3/20	1772	3/20	1786	8/20	1395	9/20	1308	6/20	1598
60	9/20	1311	7/20	1380	8/20	1389	10/20	1238	7/20	1437
80	9/20	1227	8/20	1354	11/20	1099	10/20	1320	4/20	1699
100	9/20	1265	10/20	1269	5/20	1603	13/20	999	4/20	1729
134	14/20	833	13/20	958	9/20	1243	16/20	843	3/20	1788
201	14/20	836	18/20	527	12/20	1018	18/20	607	1/20	1907
268	18/20	582	18/20	464	19/20	369	18/20	561	1/20	1919

直後の学習節の共有にかかるものを解消することができれば，Xeon Phi の多量のスレッドを有効に活用して性能向上を実現できると期待される。

5. 高速化検討

〈5·1〉 手　　法　大量の学習節を共有することによる同期待ち時間を解消する方法として，まずは共有される学習節の数を制限することが考えられる。〈2·4〉節で述べた通り，共有される学習節は，その LBD が学習節の選別時に動的に更新される閾値以下であるものに限られる。良い学習節は探索により徐々に見つかっていくことを考慮すれば，閾値は実行開始直後に大きく，実行が進むにつれて次第に小さくなると考えられる。すなわち，共有される学習節の数も実行開始直後は多いが，実行が進むにつれて少なくなる。これは〈4·3〉節の解析結果とも合致する。

以上の推察より，本稿では学習節の共有の閾値を小さな値に固定し，探索空間の削減効果が極めて高いと推測される学習節だけを共有することで，同期待ち時間の解消を図る。これは，学習節の選別における閾値の更新部分をコメントアウトすることによって実現できる。閾値は，事前の予備実験の結果を踏まえ，2〜5 に設定する。

〈5·2〉 sat3 における評価　第 2 世代 Xeon Phi において閾値を固定し，sat3 の求解を試みた場合の評価結果を Table 5 に示す。評価環境は 4 章と同様であるが，スレッド数については 40，60，80，100 スレッドに加えて，134，201，268 スレッド（Xeon Phi の物理コア数 −1 の 2，3，4 倍）でも評価を行った。また，Original は閾値を固定する前の第 2 世代 Xeon Phi の評価結果である（Table 4）。閾値を固定する前はスレッド数を大きくするほど平均実行時間が長くなったが，閾値を固定した場合はいずれの閾値でも，スレッド数を大きくするほど平均実行時間は短くなる傾向がみられた。一方，同一スレッド数の間で比較すると，特にスレッド数が 134 以下の場合には，閾値を小さくするほど平均実行時間は長くなる傾向を示した。本来ならば閾値を小さくするほど共有される学習節は少なくなり，同期待ち時間が短くなることが考えられるため，これは予想に反する結果である。

そこで，〈4·3〉節と同様にして，実時間あたりの CPU 時間を集計した。Fig. 5 に，sat3 の求解において，閾値を 5 に固定した場合のプロットを示す。80 スレッド以上において

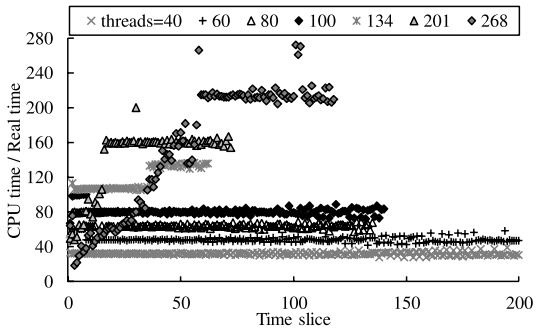


Fig. 5. Time change of CPU time per real time of sat3 on 2nd generation Xeon Phi. Threshold of LBD is set to 5.

途中でプロットが途切れているのは，そこで解が見つかったことを示す。Fig. 5 では，Fig. 4 と同様に，20 回の実行のうち単一の実行の結果をプロットしている。そのため，ここから読み取れる実行時間は Table 5 で示した平均実行時間とは必ずしも一致しないことに注意されたい。例えば，Fig. 5 において，スレッド数が 268 の場合よりも 134，201 の場合の方が早く解を見つけられている。しかし，Table 5 で示したとおり，20 回の実行の平均では 268 スレッドの場合の方が実行時間は短くなった。

スレッド数が 134 以下の場合において，実時間あたりの CPU 時間は，実行開始直後に速やかにスレッド数に近い値に到達している。このことから，〈5·1〉節で推察したとおり，閾値を小さな値に固定することで同期待ち時間をほぼ解消できていることが読み取れる。一方でこの結果は，スレッド数が 134 以下の場合に同期待ち時間を解消するには閾値は 5 で十分であることも意味している。それよりも閾値を小さくすると，今度は逆に共有する学習節が少なくなりすぎ，探索空間の削減が十分に進まないことから，平均実行時間が悪化したと考えられる。また，スレッド数が 201，268 の場合は，Fig. 4 でもみられた，スレッド数の増加に応じて実行開始直後からのプロットの傾きがゆるやかになる現象が確認できた。

〈5·3〉 全ワークロードにおける評価　〈5·2〉節の評価より，sat3 において閾値を 5 以下に固定すると，268 スレッド実行時に最も平均実行時間を短くできた（Table 5）。また，閾値を 5 としたとき，実行開始から 50 タイムスライス程度で実時間あたりの CPU 時間がスレッド数に近づ

Table 6. Evaluation with all the workload of SAT Competition 2016.

Setting	Solved	SAT	UNSAT	Time[s]	Time.solved[s]
Original	201/499	77	124	6445	992
Threshold = 2	236/499	86	150	5781	625
Threshold = 3	233/499	83	150	5741	572
Threshold = 4	229/499	88	141	5807	576
Xeon	293/499	90	203	4557	287

く，すなわち十分な並列性を引き出せることも確認できた（Fig. 5）。これにより，第 2 世代 Xeon Phi で並列実行可能な最大のスレッド数（268）で効率的に Glucose Syrup を実行できる見込みが立った。そこで最後に，SAT Competition 2016 の全ワークロードを用いた評価を行う。使用する問題は，Parallel Track に含まれる全 500 問のうち，ダウンロードできる問題セットに含まれていなかった 1 問（modgen-n200-m90860q08c40-13698.cnf）を除いた 499 問である。スレッド数は 268 とし，〈5·2〉節の結果を踏まえ，閾値を 2，3，4 に固定する場合と閾値を固定しない場合の 4 通りでの評価を行った。また，比較のため Xeon の 20 スレッド実行（閾値を固定しない）でも同様に評価した。制限時間は 5000 秒とし，平均実行時間を求める際は制限時間内に解けなかった問題の実行時間は 10000 秒とみなした。なお，閾値を 5 に固定した場合の評価は行わなかった。その理由の 1 つは，ワークロード数，制限時間の増大により，全ワークロードでの評価には多大な時間を要することである。もう 1 つは，Fig. 4 での unsat1 と sat3 の比較の結果から，他の問題では閾値をより小さくする，つまり学習節の共有をより強く制限した方が良いと考えられることである。

Table 6 に全ワークロードを用いた評価の結果を示す。Solved の列の分子は解けた問題数で，SAT，UNSAT の列は解けた問題のうち解がそれぞれ SAT，UNSAT であるものの数を示す。Time の列は平均実行時間である。また，Time.solved の列は Xeon を含む 5 つの設定すべてで解けた問題（計 180 問）における平均実行時間を示す。Xeon Phi において閾値を固定することにより，解けた問題の数は 201 から最大で 236（閾値を 2 とした場合）へと増加した。また，閾値を小さくするにつれ，解けた問題数はやや増加する傾向がみられた。一方で，平均実行時間は閾値を 3 とした場合に最小となり，全ての設定で解けた問題の平均実行時間を見ても，閾値を 2 とした場合よりも閾値を 3，4 とした場合の方が短くなった。

閾値を固定する前にも解けていた問題は，問題の探索空間自体が小さいか，共有される学習節の数が少なく同期待ち時間が元々短い問題であると推測される。こうした問題で閾値を小さくしすぎた場合，〈5·2〉節で議論したように，探索空間の削減が進まないデメリットが同期待ち時間解消のメリットを上回ったことにより，実行時間の増大に繋がったと考えられる。一方，閾値を固定する前に解けなかった問題は，探索空間がある程度大きい上に同期待ち時間が長く，並列処理が十分に行われていないことが推測される。こうした問題においては学習節の共有を強く制限することが有効に働き，それが閾値の小さい時に解ける問題数が増加する傾向に繋がったとみられる。

しかしながら，いずれの場合でも解けた問題数・平均実行時間ともに Xeon での実行結果には及ばなかった。特に解が UNSAT である問題で大きな差が見られた。その改善には共有する学習節の制限だけではもはや不十分であり，多数のスレッドに対してもスケーラビリティを保つよう，同期機構を本質的に見直す必要があることが強く示唆される。また，表には記載していないが，〈4·2〉節の第 1 世代 Xeon Phi と同様，第 2 世代 Xeon Phi での 268 スレッドの実行において計算途中でメモリ不足に陥った問題が 51 問あった。Xeon Phi は利用可能なスレッドの数に対して，利用可能なメモリはそれほど大きくない。使用されるメモリ量や共有される学習節の量などに応じて適応的に使用するスレッド数を制御することも重要だと考えられる。

6. おわりに

本稿では，並列 SAT ソルバの 1 つである Glucose Syrup に対して Xeon Phi を用いた性能評価を行い，Xeon での性能との比較を行った。その結果，実行スレッド数の増加にともなう同期待ち時間の影響で性能が低下している可能性が強く示唆された。そのため共有する学習節の制限を行うことによる同期待ち時間の解消を試み，これにより SAT Competition 2016 の全ワークロードを用いた評価において解けた問題数は，201 から 236 に増加した。しかしながら，これはなお Xeon での性能を下回る結果であった。

本稿での検討・評価により，多数のスレッド実行時のメモリ不足と同期機構のスケーラビリティの 2 つの問題が明らかになった。これらの問題を解消し，Xeon Phi などのメニーコアプロセッサで高速に動作する並列 SAT ソルバを確立することが今後の課題である。

謝　辞

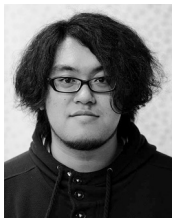
本研究の一部は JSPS 科研費 16K00072 の支援による。

文　献

（1） E. Clarke, D. Kroening, J. Ouaknine, and O. Strichma: “Computational challenges in bounded model checking”, International Journal on Software Tools for Technology Transfer, Vol.7, No.2, pp.174–183 (2005)
（2） A. Smith, A. Veneris, M.F. Ali, and A. Viglas: “Fault diagnosis and logic debugging using Boolean satisfiability”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.24, No.2, pp.1601–1621 (2005)
（3） K.H. Wang and C.M. Chan: “Incremental learning approach and SAT model for Boolean matching with don’t cares”, Proc. IEEE/ACM Conf. on Computer-Aided Design, pp.234–239, San Jose, CA, USA (2007)
（4） D. Ledoux: “An interlocking Safety Proof Applied to France Rail Network”, Proc. SAT Competition 2016, p.32, Bordeaux, France (2016)
（5） D.A. Patterson and J.L. Hennessy: “Computer Organization and Design”, 5th ed., Morgan Kaufmann (2013)
（6） G. Audemard and L. Simon: “Glucose and Syrup in the SAT Race 2015”, Proc. SAT Race 2015, Austin, TX, USA (2015)
（7） A.D. Palù, A. Dovier, A. Formisano, and E. Pontelli: “CUD@SAT: SAT

Solving on GPUs”, Journal of Experimental and Theoretical Artificial Intelligence, Vol.27, No.3, pp.293–316 (2015)
（8） T. Balyo, P. Sanders, and C. Sinz: “HordeSat: A Massively Parallel Portfolio SAT Solver”, Proc. 18th International Conference on Theory and Applications of Satisfiability Testing, pp.156–172, Austin, TX, USA (2015)
（9） S. Nishiwaki, N. Fujieda, and S. Ichikawa: “Preliminary performance evaluation of of parallel SAT solver on Xeon Phi processor”, IEEJ Technical Meeting on Innovative Industrial System, No.IIS–18–001 (2018) (in Japanese)
西脇慎太朗・藤枝直輝・市川周一：「並列 SAT ソルバの Xeon Phi プロセッサ上での予備的性能評価」, 電学次世代産業システム研, No.IIS–18–001 (2018)
（10） M. Davis, G. Logemann, and D. Loveland: “A machine program for theorem-proving”, Communications of the ACM, Vol.5, No.7, pp.394–397 (1962)
（11） J.P.M. Silva and K.A. Sakallah: “GRASP: A search algorithm for propositional satisfiability”, IEEE Transactions on Computers, Vol.48, No.5, pp.506–521 (1999)
（12） R.J. Bayardo, Jr. and R.C. Schrag: “Using CSP look-back techniques to solve real-world SAT instances”, Proc. National Conference on Artificial Intelligence, pp.203–208, Providence, RI, USA (1997)
（13） G. Audemard and L. Simon: “Predicting learnt clauses quality in modern SAT solvers”, Proc. 21st International Joint Conference on Artifical Intelligence, pp.399–404, Pasadena, CA, USA (2009)
（14） N. Eén and N. Sörensson: “MiniSat: A SAT solver with conflict-clause minimization”, Proc. SAT Competition 2005, St. Andrews, Scotland (2005)

西　脇　慎太朗（非会員） 2016 年豊橋技術科学大学電気・電子情報工学課程卒業。2018 年同大学電気・電子情報工学専攻修士課程修了。電子情報通信学会会員。



藤　枝　直　輝（非会員） 2013 年東京工業大学大学院情報理工学研究科計算工学専攻博士後期課程修了。博士（工学）。同年より豊橋技術科学大学電気・電子情報工学系助教。プロセッサアーキテクチャ，FPGA 応用，組込みシステム，セキュアプロセッサの研究に従事。情報処理学会，電子情報通信学会，IEEE 各会員。

市　川　周　一（正員） 1985 年東京大学理学部卒業。1987 年同大学大学院理学系研究科修士課程修了。1987 年新技術事業団創造科学推進事業（ERATO）後藤磁束量子情報プロジェクト研究員。1991 年三菱電機（株）LSI 研究所，システム LSI 開発研究所勤務。1994 年名古屋大学工学部助手。1997 年豊橋技術科学大学工学部講師。同助教授，准教授を経て，2010 年豊橋技術科学大学大学院工学系研究科准教授。2011 年沼津工業高等専門学校制御情報工学科教授。2012 年より，豊橋技術科学大学大学院工学系研究科教授。現在に至る。理学博士。並列計算機，並列処理，および専用計算システムアーキテクチャの研究に従事。IEEE（senior member），電子情報通信学会（シニア会員），ACM，情報処理学会，各会員。