情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-HPC-121 No.30
2009/8/6

プロファイルを使用した並列 LTL モデル検査のチューニング

小林史佳[†1]　三輪真弘[†2,*1]　上田和紀[†3]

並列 LTL モデル検査では，通信頻度や実行時の PE 数を調整することによって，その性能が大きく変わってくる．しかし，そのモデルの持つ特徴を検査前に把握し，そのモデルに最適なパラメータ設定を行うことは難しい．本論文では，実行前に短時間のプロファイルを行い，モデルに対して最適なパラメータを調べる手法について評価した．

Tuning technique of the parallel LTL model checking with profiling

Fumiyoshi Kobayashi,[†1] Masahiro Miwa [†2,*1] and Kazunori Ueda[†3]

The performance of the parallel LTL model checking is turned by tuning the communication frequency and total PEs of execution, However, it is difficult to set the optimal value for that parameters before runtime. We approach the technique of searching the optimal parameters through using the short-lasting pre-profiling.

†1 早稲田大学大学院理工学研究科
Graduate School of Science and Engineering, Waseda University
†2 早稲田大学大学院基幹理工学研究科
Graduate School of Fundamental Science and Engineering, Waseda University
†3 早稲田大学理工学術院情報理工学科
Faculty of Science and Engineering, Waseda University
*1 現在，富士通研究所
Presently with Fujitsu Laboratories LTD.

1. はじめに

モデル検査は，システムの取り得る状態空間を網羅的に探索することによって，見つけにくいバグを発見したり，システムの高信頼性を確かめることが出来るシステム検証技術である．モデル検査の問題点として，システムの規模を少し大きくするだけで状態空間が指数的に大きくなってしまうため，メモリ不足や探索の長時間化が起きてしまう問題（**状態爆発問題**）が挙げられる．この問題を解決するアプローチの 1 つである並列化の分野では，アルゴリズムの研究や実装の性能評価が行われているが，モデルの持つ特徴を考慮した実行時のチューニング手法についての研究はまだ行われていない．

我々は今回，並列モデル検査ツール DiVinE[1)] を用いて，約 1000 万〜10 億の状態を持つ 20 のモデルを対象に，通信頻度や総 PE（Processor Element）数を変えることによって生じる速度性能の変化を調査した．その結果，最適なパラメータは個々のモデルによってそれぞれ大きく異なり，一律のパラメータ設定では効率のよいモデル検査を行うことが出来ないことを明らかにした．また，実行前に短時間のプロファイルを行い，それらのパラメータを最適な値に近づけることによって，実行をより高速に行う手法を実装し評価を行った．

2. LTL モデル検査

LTL モデル検査問題は，文献 2) で述べられているオートマトンベースの手続きを用いることで，効率的に解くことができる．本章では，その手法や関連用語について記述する．

2.1 オートマトンベースの求解手続き

この手法では，まず，検査対象となるシステムを表現したシステムオートマトンと検査する性質を記述した**線形時相論理（Linear Temporal Logic）**から生成される性質オートマトンの二つの有限状態オートマトンの同期積を取る．モデルが性質を満たすかどうか判定するには，この積オートマトンが受理する実行が空であるかを決定する問題を解くことになるが，これはグラフの**受理サイクル探索問題**に帰着することができる．受理サイクル探索問題とは，**受理頂点**という特別なラベル（受理ラベル）を持つ頂点が含まれている閉路を発見する問題である．この一連の手続きを図 1 に示す．

2.2 強連結成分

頂点 u から v へ到達可能であり，同時に頂点 v から u へ到達可能であるとき，その二つの頂点は同じ**強連結成分（Strongly Connected Component）**に含まれるという．別の強連結成分に含まれている頂点同士では閉路を作ることができないため，受理サイクルは

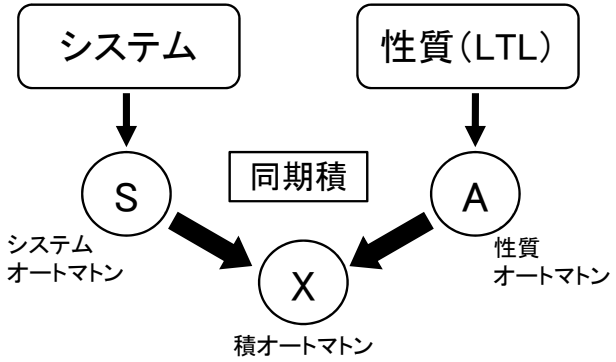

図 1 LTL モデル検査
Fig. 1 LTL model checking

同一の強連結成分に含まれた頂点集合内にしか現れない．2.1 節で述べた各オートマトンは SCC に分解することが可能であり，特に性質オートマトンは非常に小さいため，短時間で性質オートマトンを SCC に分解することが可能である．

3. 並列 LTL モデル検査

本章では，モデル検査を並列実行するために用いられる，状態空間分割手法と受理サイクル探索アルゴリズムについて記述する．

3.1 状態空間分割手法

この小節では，本論文で扱った **Hash-Based 分割法**と **Hash-on-SCC （HoS）分割法**の二つの状態空間分割手法について述べる．

効率のよい並列 LTL モデル検査を行うためには，各 PE の保持する状態を出来る限り均等に分けて，時間的・空間的な負荷を均一にすることが望ましい．その一方で，通信コストの観点から，別の PE への通信（**cross transition**）を出来る限り抑えたい．また，近年のクラスタ環境はより大規模化する傾向があるため，数百 PEs 規模の環境でも正しく動作することも重要である．これらの性質を同時に満たすような分割手法を実現するのは難しいが，ここで紹介する分割手法は，上記の性質を複数満たした効率のよい分割方法である．

- **Hash-Based 分割法**

Hash-Based 分割法は，各状態のハッシュ値を計算し，その値を用いて各 PE にランダムに状態を振り分ける手法である．Hash-Based 分割法は均等な負荷分散を実現できるという長所があるが，cross transition が増加するという短所が存在する．この分割手法は DiVinE で標準的に用いられている分割手法である．

- **Hash-on-SCC 分割法**

Hash-on-SCC 分割法は，文献 3) で新たに提案された分割方法である．2.2 節で説明した SCC 毎に複数の PE を割り当てた後，さらに Hash-Based 分割を行う．前述の通り，受理サイクルは SCC 間を跨いで存在しないため，受理サイクル探索をより局所的に行うことができる．この分割方法は Hash-Based 分割法よりも cross transition を抑えることができるものの，各 SCC に対して PE 数を適切に振り分けないと，各 PE の負荷が偏ってしまう．HoS は，標準では各 SCC に等しい PE 数を割り当てる．

3.2 探索アルゴリズム

逐次実行で最適な探索アルゴリズムとされている postorder の Nested DFS は，P- 完全とされているため，並列化は困難である．そのため，分散並列環境で代わりに動作するアルゴリズムが文献 4) で提案されている．

並列 LTL モデル検査ツール DiVinE には，この文献 4) で提案された受理サイクル探索アルゴリズムが複数実装されている．これらの探索アルゴリズムのうち，本論文で使用したアルゴリズムである **OWCTY （One-Way-Catch-Them-Young）**について説明する．

OWCTY アルゴリズムは，受理サイクル探索問題を「オートマトングラフが自明でない受理強連結成分を含んでいるか否か」という問題に捉えなおして求める手法である．このアルゴリズムの疑似コードを図 2 に示す．このアルゴリズムはまず，到達可能な全ての頂点を求めたのち，次の二つの規則に基づいて非受理成分の頂点を取り除いていく．

- 頂点から到達可能な受理頂点が存在しない場合，その頂点はどの受理成分にも属さない．
- 頂点から進む辺が存在しない場合，その頂点はどの受理成分にも属さない．

この作業を繰り返して不動点に到達したとき，残った頂点集合が空である場合は受理サイクルが存在せず，空集合でない場合は受理サイクルが存在する．

OWCTY アルゴリズムは on-the-fly には動作しないため，受理サイクルが存在するモデルに対してはあまり効率が良くないが，受理サイクルが存在しないモデルに対しては高速，かつ安定した性能を示すことが文献 5) による性能評価実験によって示されている．

4. 実験に使用したモデル，および実験環境

5 章，6 章での実験に用いた環境の詳細を，**表 1** にまとめた．また，実験で用いたモデル

```
proc SCC((V, E, s, A))
    S := Reachability(A);
    old := ∅;
    while S ≠ old do
        S := Reachability(S ∩ A)
        S := Elimination(S)
    od
    if S ≠ ∅
        then report (NO ACCEPTING CYCLE exists)
        else report (ACCEPTING CYCLE found)
    fi
end
```

図 2 OWCTY Algorithm
Fig. 2 OWCTY Algorithm

は，モデル検査プログラムのベンチマーク BEEM[6] から，状態数が約 1000 万〜10 億を持つ 20 のモデルを選んで使用した．モデルの詳細を**表 2** に記す．表 2 の各データは次のような意味である．

- **name**：モデルの名称を
- **result**：モデル検査の結果（valid：受理サイクルなし，invalid：受理サイクルあり）
- **scc**：性質オートマトンの強連結成分数
- **states**：モデルの状態数
- **transitions**：モデルの遷移数
- **time**：128PEs を使用したときの探索時間（単位：秒）

5. チューニング

1 章で述べたとおり，並列 LTL モデル検査におけるパラメータチューニングに関する研究事例はなく，どのようなパラメータが並列 LTL モデル検査にとって効果があるのか定かではない．そこでまず，次の三つのパラメータについて，その最適値と速度向上効果を調査し，探索実行時間を最適化する実験を行った．

- HoS 分割法における SCC 毎の PE 数の割当て

表 1 実験環境
Table 1 experimentation environment

ノード数	28
機種	Sun Fire X4600 M2
CPU	AMD Opteron 8222 3GHz Dual Core × 8
NIC	Sun X1027A-Z Dual 10 Gigabit Ethernet PCI-E x8
主記憶	256GB
ハードディスク	146GB × 4
OS	Debian/GNU Linux 5.0 (Lenny)，Kernel 2.6.26

表 2 実験用モデル
Table 2 experimental models

Name	result	scc	states	transitions	time（sec）
anderson.8.prop4	valid	2	1071370231	7499411024	1167.42
elevator.5.prop3	valid	2	205646803	709182775	164.3
elevator2.3.prop4	valid	6	14979042	204078365	51.83
lamport.7.prop4	valid	2	74413141	423216967	96.14
leader_election.6.prop2	valid	1	35773430	233181690	110.66
leader_filters.6.prop2	valid	1	212652416	551025365	148.92
mcs.5.prop4	valid	2	119663657	693157106	152.17
peterson.7.prop4	valid	2	284942015	1835563756	364.02
public_subscribe.5.prop1	valid	2	1242073745	5804882971	979.03
synapse.7.prop3	valid	4	15373745	31605256	21.47
anderson.6.prop3	invalid	2	36381283	254109048	77.97
brp.5.prop2	invalid	2	35046929	108643444	100.52
elevator.5.prop2	invalid	2	274348677	1039091137	568.41
iprotocol.6.prop4	invalid	3	104232788	385849887	465.26
lamport.7.prop2	invalid	2	72511607	312348399	116.8
lifts.8.prop4	invalid	2	19531231	55196308	58.43
phils.8.prop3	invalid	2	61230190	571951235	162.11
plc.4.prop1	invalid	3	8547938	21602807	82.32
resistance.2.prop4	invalid	2	107748682	563921013	1061.49
train-gate.7.prop2	invalid	2	90009670	194260275	178.17

- ネットワークバッファサイズの上限値
- 実行時の総 PE 数

5.1 HoS 分割法における SCC 毎の PE 数の割当て

HoS 分割法は各 SCC に対して均等に PE を割り当てている．しかし，**表 3** に示すように，各 SCC が処理する状態数が不均一になるモデルも存在する．なお，今回扱ったモデル

表 3 各モデルにおける SCC 毎の状態数
Table 3 The number of states per SCC in each models

Name	scc 0	scc 1	scc 2	scc 3	scc 4	scc 5
elevator.5.prop3	185008051	20638752	—	—	—	—
elevator2.3.prop4	7667712	3997670	679934	2625535	8191	0
public_subscribe.5.prop1	1153014089	89059656	—	—	—	—
iprotocol.6.prop4	1	41387484	62845303	—	—	—
phils.8.prop3.dve	43046720	18183470	—	—	—	—
plc.4.prop1	3763999	3669469	1114470	—	—	—

の中で，各 SCC に含まれる状態数がほぼ均等になるモデルは 20 モデル中 8 モデルだった．

各 SCC の状態数は一度モデル検査を実行してみないと分からないため，一度検査を実行して各モデルにおける SCC 毎の状態数を調べた後，その結果をもとに各 PE が処理する状態数が均等になるように PE 数を割り当てて実行させた．その結果，特に実行時間の変化が激しかった 10 のモデルについて，図 3 に示す．

3 本の棒グラフのうち，左が DiVinE 標準の Hash-Based 分割を用いた場合の性能で，他の棒グラフの基準値となっている．中央が最適化前の HoS 分割法を使用した場合の性能向上比，右が最適化後の性能向上比となっている．最適化前に性能が落ちていたモデルであっても，最適化を行うと多くのモデルで Hash-based 分割法と同程度の性能まで戻っているものの，最適化前の性能が著しく悪い場合は最適化を行っても Hash-based 分割法より性能が落ちる結果となった．

5.2 ネットワークバッファサイズの上限値

DiVinE では，通信を行う際に用いるバッファサイズの上限値が 8192byte となっている．この値を大きくすると一度に多くのデータを通信できるため，通信コストを抑えることができる．その反面，通信頻度が減少して通信待ちが大きくなる可能性が高まる．また，バッファサイズを増やしていくと，それに伴ってメモリ使用量が多くなっていき，メモリ不足が起きやすくなるため，単純に大きくすればいいというわけではない．この実験では，ネットワークバッファサイズの上限値を標準の 8192byte から変化させていき，各モデルについて 2 つの分割法を使用した場合の最適なネットワークバッファサイズを求めた．HoS 分割法は 5.1 節で最適化した PE 数の割当てを用いている．一部のモデルについての Hash-Based 分割法の結果を図 4 に，HoS 分割法の結果を図 5 に示す．

いずれの分割法でも，最適なネットワークバッファの値はモデルによって大きく異なっていた．半数近くのモデルがバッファサイズを大きくすると，それに伴って高速に探索できるようになっていたが，バッファサイズを大きくしすぎると却って速度が減少してしまうことが多かった．また，求められた最適なネットワークバッファの値はモデルの取りうる状態空間の大きさや性質オートマトンとは無関係な値となっており，モデル検査の実行前にモデルを静的に分析して最適な値を求めることは難しいと考えられる．

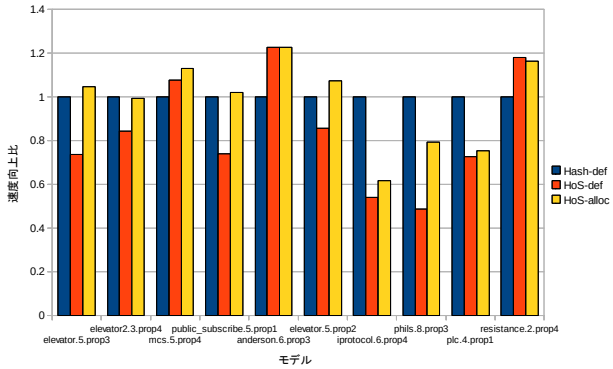

図 3 SCC 毎に最適な PE 数を割り当てた場合の性能向上比
Fig. 3 The speedup assigned the optimal number of PEs per SCC

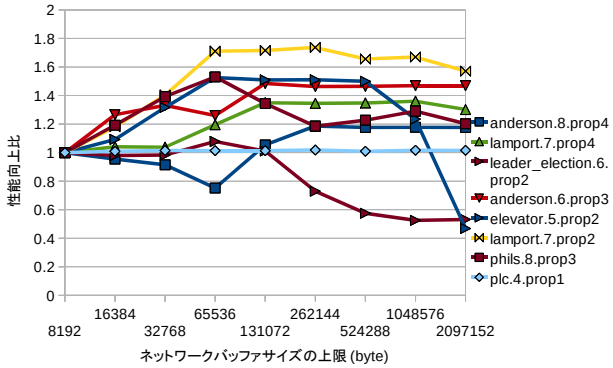

図 4 ネットワークバッファサイズの上限値と性能向上比（Hash-Based 分割法）
Fig. 4 The effect of tuning Network-Buffer-Size-Limit (Hash-Based partition)

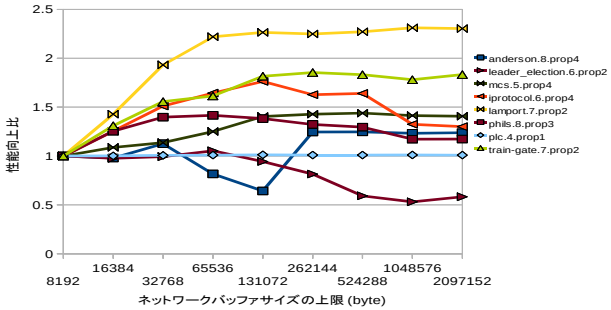

図 5 ネットワークバッファサイズの上限値と性能向上比（HoS 分割法）
Fig. 5 The effect of tuning Network-Buffer-Size-Limit (HoS partition)

5.3 実行時の総 PE 数

一般的に，並列化手法では PE の数を増やすほど性能が上昇するが，ある PE 数を越えると通信等の並列化によって増加するコストがオーバーヘッドとなって性能が低下してしまう．今回，実験に使用したクラスタマシンは，各マシンが Quad core×4 のマルチコアマシンであり，1 つのマシンで 16PE を立ち上げてしまうと，通信頻度によってはネットワークの通信で衝突が起きてしまい，却って性能が低下してしまっている可能性が考えられる．

上記のような点を考えて，一つのマシンで立ち上げる PE の数を 6 から 16 まで変化させ，その速度の違いについてまとめたところ，Hash-Based 分割法では図 6 のような結果となり，HoS 分割法では図 7 のような結果となった．なお，各モデルのネットワークバッファサイズの値は 5.2 の実験結果を反映した最適な値を用いている．

この実験では，モデルの取りうる状態空間の規模と性能変化に若干の関連性が見られた．規模の大きいモデルは多めの PE を使った方が高速に解くことが多く，逆にそれほど規模の大きくないモデルに関しては少なめに PE を使った方が高速に解けていた．しかし，一部のモデルでは総 PE 数を 80 にすると著しく性能が低下してしまうようなことが起こることもあった．

5.4 チューニング結果

今までに行ったチューニングによって，最終的に図 8 のような速度向上が得られた．

この図 8 に載せたモデルは特に速度向上率が高かったものであり，標準のパラメータを用いた Hash-Based 分割法と比べると，Hash-Based 分割法では最大で 2.28 倍の速度向上となり，HoS 分割法では 2.41 倍の速度向上となった．速度向上が最も小さいモデルであっても Hash-Based 分割法で 1.27 倍，HoS 分割法では 1.08 倍の速度向上となり，実行時にこれらのパラメータに対して最適な値を与えることで，並列 LTL モデル検査を高速に行うことができることが保証された．

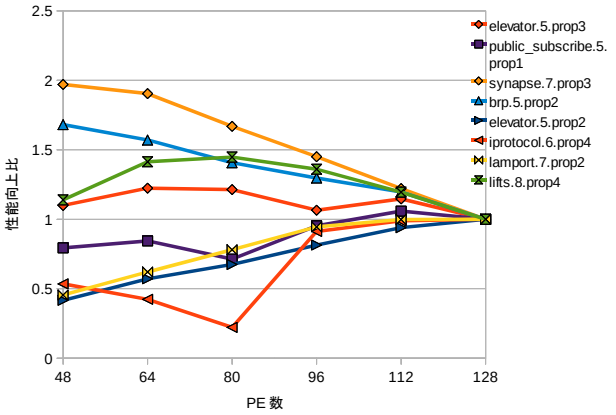

図 6 PE 数の変化と性能向上比（Hash-Based 分割法）
Fig. 6 The effect of tuning the number of PEs (Hash-Based partition)

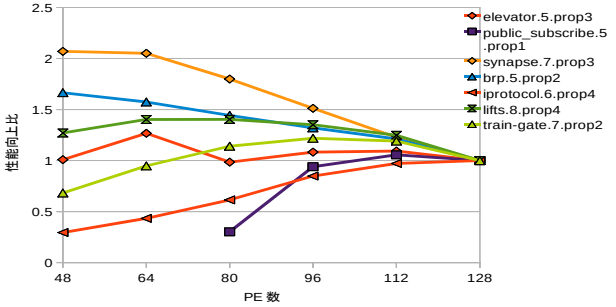

図 7 PE 数の変化と性能向上比（HoS 分割法）
Fig. 7 The effect of tuning the number of PEs (HoS partition)

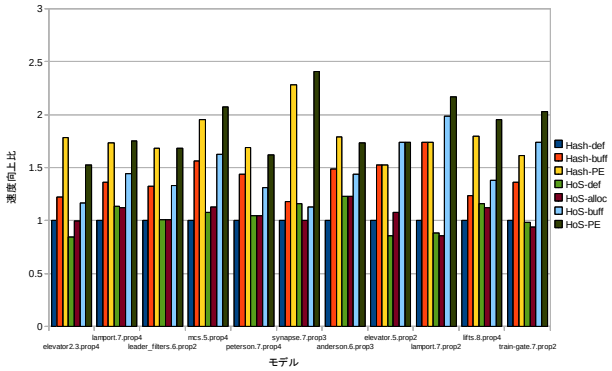

図 8 各モデルにおける最終的な速度向上比
Fig. 8 The final speedup in each models

6. プロファイル

5 節の結果から，モデル検査実行時に各パラメータを適切に与えることで，実行時間の短縮を図ることができると考えられる．しかし，これらのパラメータの最適値を実行前に予測をすることは非常に難しい．

そこで，実際のモデル検査実行前に短時間の**事前プロファイル**を行うことによって，これらの最適値に近い値を予測するプログラムを実装し，評価を行った．

表 4 は，peterson.7.prop4 モデルに対して行ったプロファイル結果である．この実験では 5.2 節と同じく，ネットワークバッファサイズを標準の 8192 から 2097152 から変化させ，10 秒間の到達性検査を行い，状態数を調べたものである．peterson モデルでは，256KB である時が最も性能が良い結果が出ていたが，プロファイル結果では 16KB が最も状態数を得る結果となってしまっている．他のモデルでも同じような結果が出てしまっていることから，モデル検査の開始直後は通信頻度が安定しておらず，バッファサイズが小さいほど通信が頻繁に行われ，ある程度の状態数に安定して到達可能であることが原因だと予測される．

7. まとめと今後の課題

これまでの実験結果から，各パラメータを適切な値に調整し実行することで，その速度性能を大きく上昇させることができることが判明した．その一方で，モデル検査開始直後の短時間の到達性検査では安定して最適なパラメータの値を反映するのは難しく，この手法を用いて効果を出すには非常に時間のかかる問題に対して，ある程度長い時間のプロファイルを取る必要があるだろう．

表 4 peterson モデルにおける短時間プロファイルで得られた状態数
Table 4 The number of states of the peterson Model with pre-profiling

バッファサイズ	1 回目	2 回目	3 回目	4 回目	5 回目	平均値	比率
8192	1961711	321258	1538486	1965704	717664	1300964.6	1
16384	1733723	1199696	2239419	1868260	1400879	1688395.4	1.3
32768	1369214	221665	1015641	1258369	1650097	1102997.2	0.85
65536	730914	1239983	941791	731740	372219	803329.4	0.62
131072	1210354	1981981	1554816	603791	264133	1123015	0.86
262144	278579	1245526	1493763	1856478	1438400	1262549.2	0.97
524288	2326260	1121613	1385889	772372	328646	1186956	0.91
1048576	734437	495975	1696837	521614	925940	874960.6	0.67
2097152	1759319	1046281	2027303	185993	355952	1074969.6	0.83

今後の課題としては，到達性検査で得られるデータのバリエーションを増やすことで，よりプロファイルの精度を増すことが考えられる．現在は，到達可能な状態を調べることでのみ最適な値を算出していたが，それ以外のデータを有効に活用することで，より効果的なプロファイルを行うことが可能ではないかと考えられる．

謝辞 本研究では，独立行政法人産業技術総合研究所連携検証施設さつきの検証クラスタを利用しました．We acknowledge our use of verification clusters in SATSUKI, Collaborative Facilities for Verification, AIST.

参 考 文 献

1) Barnat, J., Brim, L., Černá, I., Moravec, P., Ročkai, P. and Šimeček, P.: DiVinE – A Tool for Distributed Verification (Tool Paper), *Computer Aided Verification*, LNCS, Vol.4144/2006, Springer Berlin / Heidelberg, pp.278–281 (2006).
2) Moshe Y. Vardi, and PierreWolper: An automata-theoretic approach to automatic program verification. In Proc. First IEEE Symp. on Logic in Computer Science, 1986, pp. 322–331.
3) 三輪真弘，上田和紀：強連結成分ベースのグラフ分割による分散並列 LTL モデル検査の高速化，情報処理学会第 71 回全国大会，分冊 1, pp.25–26 (2009).
4) Barnat, J., Brim, L. and Černá, I.: Cluster-Based LTL Model Checking of Large Systems, *Formal Methods for Components and Objects*, LNCS, No.4111, pp.259–279 (2005).

Vol.2009-HPC-121 No.30
2009/8/6

5) 渋谷健介，上田和紀：分散検証環境 DiVinE を用いた分散 LTL モデル検査アルゴリズムの性能評価，情報処理学会第 70 回全国大会.
6) Pelánek, R.: BEEM: Benchmarks for Explicit Model Checkers, *Proc. of SPIN Workshop*, LNCS, Vol.4595, Springer, pp.263–267 (2007).