大規模メモリ環境下におけるモデル検査ツール Spin のマルチコア検証機能の性能評価

小林 史佳 †　　上田 和紀 ‡
† 早稲田大学理工学研究科　　‡ 早稲田大学理工学術院

1 研究の背景と目的

モデル検査はグラフ探索問題に帰着して解くことが出来るが，対象システムが大きくなると探索空間が爆発的に増大し探索時間が増加する．モデル検査ツール Spin はその対策としてマルチコア環境用の並列モデル検査機能を実装した [1] が，大規模メモリのマシン上で性能を評価した報告は無い．そこで本研究では，大規模メモリ環境下での Spin の性能を調査することを目的とする．

2 モデル検査ツール Spin

Spin [2] は非同期並行システムの検証に広く使用されているモデル検査ツールである．検査できる性質は安全性と活性の 2 つがあり，別々の手法で検査する．Spin はモデル検査を高速に行うために，on-the-fly による探索グラフの構築と Partial Order Reduction によるグラフの簡約化という 2 つの技術を実装している．

2.1 安全性の検査

安全性の検査は，グラフの到達性問題に帰着して解くことができる．Spin は Depth First Search (DFS) を用いて頂点の探索を行っている．

並列アルゴリズム　N 個の各コアに 0〜$N-1$ の ID を振る．cpu_0 は初期頂点から決められた深さ z までを DFS で探索していき，深さ $z+1$ に頂点が存在する場合，その頂点を cpu_1 に渡す．以降，cpu_0 と同様に，cpu_i は渡された頂点を初期頂点として深さ z までを探索し，深さ $z+1$ にある頂点を cpu_{i+1} に渡す．cpu_{N-1} は cpu_0 へと頂点を渡すことで，深さ $z \times N$ 以降の探索を行う．

2.2 活性の検査

活性の検査はグラフの閉路発見問題に帰着して解くことができる．Spin は Nested Depth First Search (Nested DFS) を用いて探索を行っている．Nested DFS は，初期頂点から到達可能な受理頂点を発見し，その受理状態から 2 度目の DFS を開始して，同じ受理頂点へ戻れるか調べることで閉路を発見する．

並列アルゴリズム　Nested DFS の 1 度目の探索と 2 度目の探索を，それぞれ別のコアが担当することで並列化を実現しているため，使用可能なコア数に 2 コアまでという制限がある．実行コアを cpu_0, cpu_1 とすると，まず，cpu_0 が 1 度目の探索を行って受理頂点を探索していき，発見した受理頂点を cpu_1 の持つキューに入れる．cpu_1 はキューに入っている頂点を取り出し，その頂点を初期頂点として 2 度目の探索を始めて，同じ受理頂点へ至る閉路が有るかを探索する．

3 評価実験

実験は，Spin ver.5.1.6 を用いて，2 つの異なる実験環境で行った．これらの環境の詳細は表 1 に示す．この表のメモリの読み書き速度の項目は実測に基づいた値であり，metton はコアとメモリの位置関係から 2 種類の数値が現れたので，その両方の値を記載している．使用するモデルは BEEM ベンチマーク [3] にあるモデルのうち，Spin で利用可能なモデル 235 個を用いた．これらのモデルのうち，metton で 40 モデル，trappe で 15 モデルがメモリオーバーとなり，完全に探索することが出来なかった．探索できた状態数の最大値は，metton では 2.53E+8 状態で，trappe では 1.56E+9 状態となり，探索時間はそれぞれ 1280 秒，17500 秒だった．

表 1: 実験環境

マシン名	metton	trappe
CPU	AMD Opteron(tm)	Intel(R) Xeon(R)
CPU 周波数	2.3GHz	2.9GHz
コア数 (全コア数)	Quad-core × 2 (8)	Quad core × 4 (16)
メモリ容量	16 Gbyte	128 Gbyte
メモリ読み込み速度	3.29E+10 bps 2.26E+10 bps	1.91E+10 bps
メモリ書き込み速度	2.44E+10 bps 1.82E+10 bps	1.03E+10 bps

3.1 エラーがない場合の安全性検査における速度調査

エラーがないモデルについて，1 コアとコア数毎の速度向上比を調査したところ，1 コアでの探索が 10 秒以上かかるような，ある程度規模の大きいモデルに対しては並列化の効果が出た．1 コアでの実行が 10 秒以上かかるモデルについて速度向上比をまとめると，図 1 と図 2 のようになった．

Performance evaluation of the model checker Spin multi-core options on the large-scale memory environment
†Fumiyoshi KOBAYASHI　‡Kazunori UEDA
†Graduate School of Science and Engineering, Waseda University
‡Faculty of Science and Engineering, Waseda University

metton の場合，8 コアでは規模の大きなモデルに対しては概ね 3〜5 倍の速度向上が得られ，最大で 6.42 倍の速度向上があった．一方，trappe ではコア数が増えても速度は全体的に上昇せず，16 コアで最大 2.25 倍の速度向上が得られたものの，かなり規模の大きなモデルでないと速度が低下しており，4 コア，8 コアの時の方が速度が向上したモデルが多く見られた．

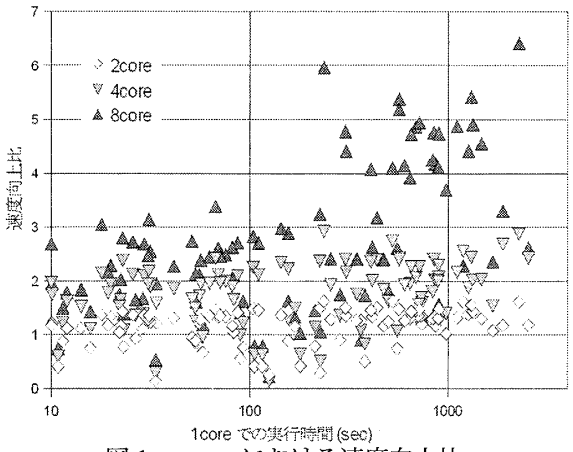

図 1: metton における速度向上比

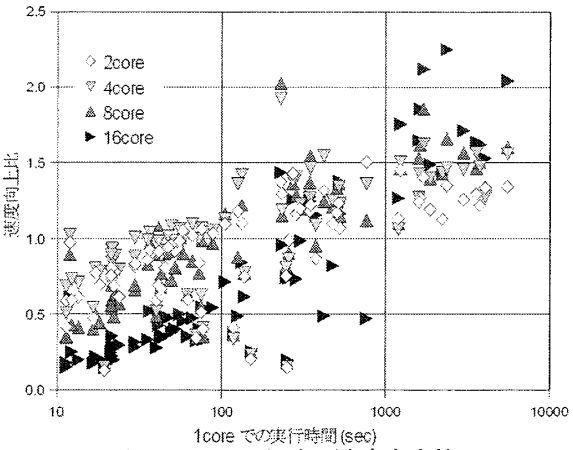

図 2: trappe における速度向上比

3.2 エラーがない場合の活性検査における速度調査

活性検査では安全性検査以上に並列効果が出にくく，1 コアでの探索に 100 秒以上かかる大きなモデルであれば，metton では約 1.1 倍〜1.4 倍程度の速度向上が見られ，1.8 倍弱速度が向上したモデルが存在した．trappe ではほぼすべてのモデルで速度が低下したが，1.1 倍弱速度が向上したモデルもあった．

3.3 エラーが有る場合の速度調査

安全性，活性のそれぞれの性質についてエラー発見までの速度を比較したところ，ほとんどのモデルで速度は低下したが，各マシンでメモリオーバーとなるほど大規模なモデルの場合，1 コアでは発見できなかったエラーがマルチコアでは発見できたケースも存在した．

3.4 metton と trappe の探索速度の比較

metton と trappe の探索速度を，実行時間が 10 秒を越えるモデルについて比較した．1 コアでは trappe の方が 1.2 倍〜1.5 倍程度の速度で探索できていたが，2 コアでは metton の方が高速なモデルが多くなり，平均 1.2 倍の速度で探索できた．4 コアでは metton の方が時間のかかるモデルは無くなり，概ね 1.5 倍〜2 倍の速度で探索するようになり，8 コアでは多くのモデルで 2 倍〜4 倍程度の速度で探索でき，最大 4.82 倍の速度比となったモデルもあった．

4 考察とまとめ

実行環境によって並列効果は大きく変わり，CPU 周波数が高く，ハッシュテーブルサイズを大きく取った trappe よりも，metton の方が並列効果が出ており，コア数が増えると実際の実行時間でも metton の方が速くなるモデルも多く存在した．モデル検査では大量の状態をメモリに記憶して，その状態の走査を行う必要があるため，メモリアクセスが頻繁に起きる．そのため，並列に探索を行う場合は CPU の処理速度以上にメモリへのアクセス速度が速い方が性能が向上しやすいと考えられる．

Spin は on-the-fly で動作するため，エラーがある場合の実験では 1 コアでもほとんどのモデルで 10 秒以内にエラーを発見してしまっており，並列効果の出やすい，大規模のグラフ探索にならないため速度向上が得られない．ただし，深い位置にエラーが存在しているモデルの場合，マルチコアでのみエラーを発見できたケースがあり，これは，並列アルゴリズムを用いることで探索の順序が変わったものと考えられる．この逆のパターンが起こり，複数コアを使った場合，大きく性能が低下してしまうケースも確認された．

活性検査の並列アルゴリズムは最大で 2 コアまでしか実行できないため，スケーラビリティという点で改善の余地がある．文献 [4] では並列アルゴリズムがいくつか提案されており，これらのアルゴリズムをマルチコアマシン上で効率的に動作するように実装し，評価することは今後の課題となる．

参考文献

[1] G. J. Holzmann and D. Bosnacki. The Design of a Multi-core Extension of the SPIN Model Checker. IEEE Trans. on Software Engineering vol.33, pp.659–674, 2007.

[2] G. J. Holzmann. The Spin Model Checker, Primer and Reference Manual. Addison-Wesley, 2003.

[3] R. Pelanek. Web portal for benchmarking explicit model checkers. Tech. Report FIMU-RS-2006-03, Masaryk University Brno, 2006.

[4] J. Barnat, L. Brim, and I. Cerna. Cluster-Based LTL Model Checking of Large Systems. In Formal Methods for Components and Objects, pp.259–279, 2005.