並列タブロー法の負荷分散について

Load sharing for parallel tableau expantion

比嘉 薫† 河野 真治††
Kaoru Higa Shinji Kono

† 琉球大学理工学研究科情報工学専攻
Specialty of Information Engineering, University of the Ryukyus,
†† 琉球大学工学部情報工学科
科学技術振興事業団さきがけ研究 21(機能と構成)
Information Engineering, University of the Ryukyus,
PRESTO, Japan Science and Technology Corporation

時相論理検証をタブロー法を状態空間の分割を用いて並列処理する方法の実装を行っている. 検証が進むにつれて状態空間が各プロセッサに不均一に分布してしまい, 一部の少数の状態の展開が少数のプロセッサ上で長く続いてしまう. これにより, 並列実装の効率が 1/2 から 1/5 程度に落ちてしまう. この不均一性は, 状態空間を分割するハッシュ関数と証明する時相論理式に依存する. ここでは, 分割の基準となるハッシュ関数の性質を調べた. また, 時相論理式の特徴を利用した状態空間の先読みによる均一性の改善法について考察する.

ソフトウェアやハードウェアの検証を行う方法と有効な手法として時相論理が挙げられる. 本稿では時区間時相論理 (Interval Temporal Logic) にタブロー法を用いる. この検証方法は, 論理式をタブローのルールに従って展開していき, 新しい状態を表す論理式 (状態式) が生成されなくなった時点で終了する. 生成される状態式の数は最悪, 部分項の組合せであり, 状態遷移数は最悪, 式中の変数の数の指数乗となる. 検証の限界は生成される状態式の数で決まる.

1 時区間時相論理 (Interval Temporal Logic)

ITL は時区間によって論理式の真理値が異なる論理である. この論理の意味論の定義には,M(p): meaning function を使う. これによって式の真理値を決めることができる.σ_0 は時刻を表し, $\sigma_0 \ldots \sigma_n$ は時区間を表す. 論理式の値は時区間に依存して決まるが, 命題変数の値は時刻だけに依存して決まる (local 性). $P\&Q$ は,P と Q が続いて起きることを表し,$@P$ は, 次の時刻に P が起きることを表している.

2 並列タブロー法

ある状態を表す時相論理式から次の状態への遷移の数は, その状態式が含む変数の指数乗存在する. 一方で, 生成される状態式は, 同じものが繰り返し現れることが多い. 指数乗の状態遷移を計算することに大量の計算力が必要だが, タブロー展開を終了できるかどうかは, 多量の状態式を格納できるかどうかによって決まる. そこで, 状態式を多数の計算機に分散して格納し, 状態遷移を並列に計算する手法が有効であると考えられる. このようにすれば, 状態式のやりとりの通信量に比べて, 状態遷移を計算する計算量の方が大きくなるので,PC クラスタ向きの計算となる.

3 状態式の表現

状態式は一般に複雑な時相論理式となるが,Binary Subterm Diagram を用いることにより, 比較的コンパクトな表現とすることができる. Binary Subterm Diagram はもとの時相論理式の部分項を条件とする二分木である. ここでは,$(Condition, True, False)$ という形で二分木を表す. ここで Condition は変数, または時相論理演算子を先頭に持つ部分項である. True,False は Condition が真, 偽の時に対応する二分木である.

例えば ◇□ p は, まず & を用いて $T\&\neg(T\&\neg p)$ に変換される. ここで $\neg p$ を $?(p, F, T)$ に変換し,$s_1 = T\&?(p, F, T)$ とすると, 元の式を $s_2 = T\&?(s_1, F, T)$ と表現できる. この式を展開すると $\neg(T\&\neg p) \vee (T\&\neg(T\&\neg p))$ が生成される. この式は複雑に見えるが Binary Subterm Tree に変換すると $s_3 = ?(s_2, T, ?(s_1, F, T))$ と単純になる (図 1).

$M_{\sigma_0\sigma_1\ldots\sigma_n}(P)$	=	$M_{\sigma_0\sigma_1\ldots\sigma_n}(p) = M_{\sigma_0}(P)$ P は命題変数
$M_{\sigma_0\sigma_1\ldots\sigma_n}(P\&Q)$	=	$0 \leq \exists i \leq n \wedge\ M_{\sigma_0\ldots\sigma_i}(P) \wedge M_{\sigma_i\ldots\sigma_n}(Q)$ (P,Q は任意の式)
$M_{\sigma_0\sigma_1\ldots\sigma_n}(@P)$	=	$M_{\sigma_1\ldots\sigma_n}(P)$ もし n=0 なら false
$M_{\sigma_0\sigma_1\ldots\sigma_n}(p)$	=	true iff n = 0

表 1: ITL の定義

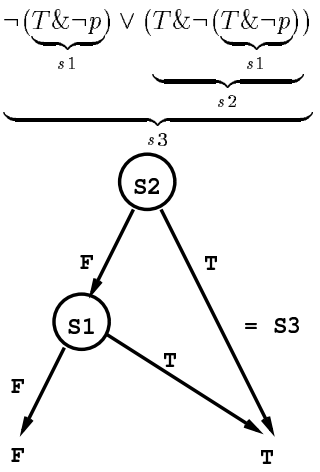

図 1: Binary Subterm Tree

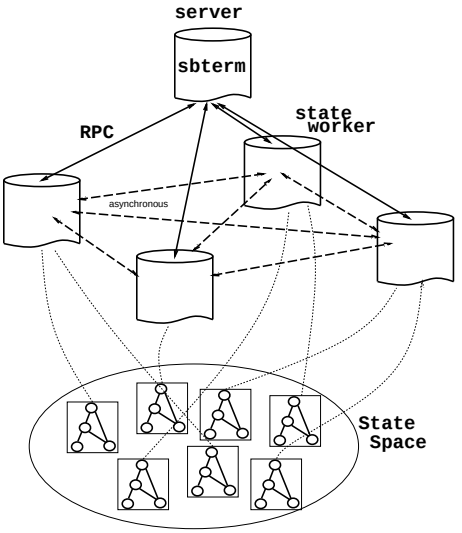

図 2: タブロー法の分散計算

部分項は有限ではあるが, 展開時に生成されることもある. 部分項の順序を固定し, 木を最大限に共有することで Binary Subterm Diagram を構成する. したがって,ITL の状態空間は部分項と,Binary Subterm Diagram の二つの要素から構成される.

検証を並列に実行するために, 状態式の展開により生成された状態式の集合を分割し, その展開処理をデータ並列的に行う. 分割は、Binary Subterm Diagram にハッシュ関数を定義することで行う. ただし、部分項は共有されているものとする. 部分項を共有しないと, Binary Subterm Diagram の形の一意性を保証することはできない. 部分項の共有は, 部分項に unique ID を割り当てることに相当する. (図 2)

4 状態式に対するハッシュ関数の影響

並列検証 [3] を行う時, 検証が進むにつれて状態空間が各プロセッサ上に不均一に分布してしまう. その理由に一部少数のプロセッサ上で少数の状態が長く展開されているためである. ここでは, 分割の基準となるハッシュ関数の改善と, 状態空間の先読みによる効率向上手法について考察する.

ここでは「6 人の食事をする哲学者問題」の検証を行った時の各計算機における各ハッシュ関数における負荷分散を調べた.

4.1 ハッシュ関数

展開により新たに生成された状態式の集合を, 複数存在する展開サーバ間で均等に振り分けるために,Binary Subterm Diagram にハッシュ関数を定義する. なお展開サーバにはそれぞれ Unique な番号が割り振られているものとする.

ここではハッシュ関数として以下の式を用いた.

$hash(true) = 7$

$hash(false) = 4$

$hash(?(C,T,F)) =$
$num(C) + hash(T) *$ 定数 $1 - hash(F) *$ 定数 2

$num(C)$ は、部分項に割り当てられた Unique ID である. このハッシュ値の計算機の台数分の剰余をとり, その値にしたがって各展開サーバに論理式を分割していく.

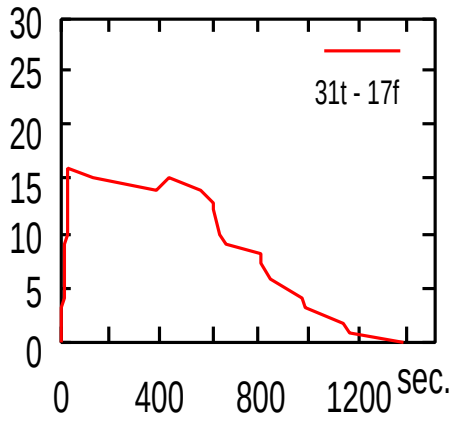

図 4: 5 階建てのエレベータ問題

5 ハッシュ関数の評価

今回は上述したハッシュ関数に対して, いくつかの定数を用いてそれぞれのハッシュ関数における計算機の負荷分散の計測を行った.(図 3) それぞれのグラフは縦軸を稼働計算機の数 (台), 横軸を時間 (sec.) を示している.

プログラムは SICStus prolog を用い, それぞれ 800MHz,512MB の RAM を持つ 31 台の PC クラスタを使って評価を行った. また通信ライブラリとして本研究室で作成した N 対 N の完全結合接続が可能な Datagram Library[4] を使った.

図 3 のグラフから,31t-19f や 13t-f の場合は, 状態式がうまく PC クラスタに均一に割り振られず, 少数の計算機上でのみ計算が進むようになってしまうのがわかる. しかし,31t-17f の場合は, 稼働率が高いまま, 一気に計算が終了する. これは, 状態式の分割がうまくいっているからである.

この 6 人の哲学者のような問題では, 状態式は,6 人の状態の組合せがすべて出て来ると考えられる. この対称性があるために, きちんとしたハッシュ関数を用いれば, 並列タブロー計算の台数効果が出ることになる.

状態の対称性が明示的にわかる場合には, その対称性を利用して状態空間そのものを縮小することができる. しかし対称性が自明でない場合も多くあり, また, 部分的な対称性などは検出すること自体も難しい. このような場合に,PC クラスタを使った直接的な検証が有効であると考えられる.

図 4 は最もよく稼働していたハッシュ関数 (31t-17f) を用いて 5 階のリフト問題の検証を行った結果である. このグラフでは稼働台数は 15 台以上にはならず状態式の展開が少数のプロセッサ上で長く展開の変数依存が少ない. これは哲学者の問題においては各状態における変数の組合せが少なく, 高速に状態の組合せを作れるのに対してリフト問題では変数の組合せが哲学者の問題と比べて多い. その結果として新しい状態空間の生成がほとんど行われず, 状態空間の展開が先延ばしにされているためである.

5.1 並列検証の効率化

現状の方法では, 展開を行ううえで, 一部の状態式の展開が先伸ばしされてしまう. そこで, この先読みのアルゴリズムを用いて先延ばしにされるであろう状態式の特性を求め, それを稼働していない計算機上で式の展開を行うことが目的でなる.

5.2 並列検証の効率化

並列タブロー法を行う場合, 先延びになった状態式をいかに速く展開するかもまた問題になる. また, そこで新しい状態式がでる度にハッシュ関数によって別の計算機へ送信される. この場合は, 稼働している計算機の台数が少なった場合, 残り少なくなった各計算機上で展開を行うほうが, 送受信にかかる時間も軽減することが可能である。そこで並列タブロー法を効率化するために以下の手順に従って並列タブローの実装を行った.

1. 並列タブローにおける計算機の最小稼働台数をあらかじめ決める
2. 並列タブロー法によって式の展開を行う
3. ある程度の大きな式の展開が終了することで, 計算機の稼働台数も徐々に減る
4. 新しい状態式が出てきた場合, それを他の計算機に転送せずに, 改めてその新しい状態式の展開を同じ計算機上で行う
5. 全ての状態式の展開が終ったところで, それぞれの計算機のキューの長さを調べる. 全てのキューの長さが 0 であれば, 検証を終了する. 0 でない場合は改めてキューの中身を取り出し, 検証を行う.

6 先読みアルゴリズムの提案

上述したハッシュ関数のみでは論理式によって, 最後の状態式が延びてしまい並列検証としては望まし

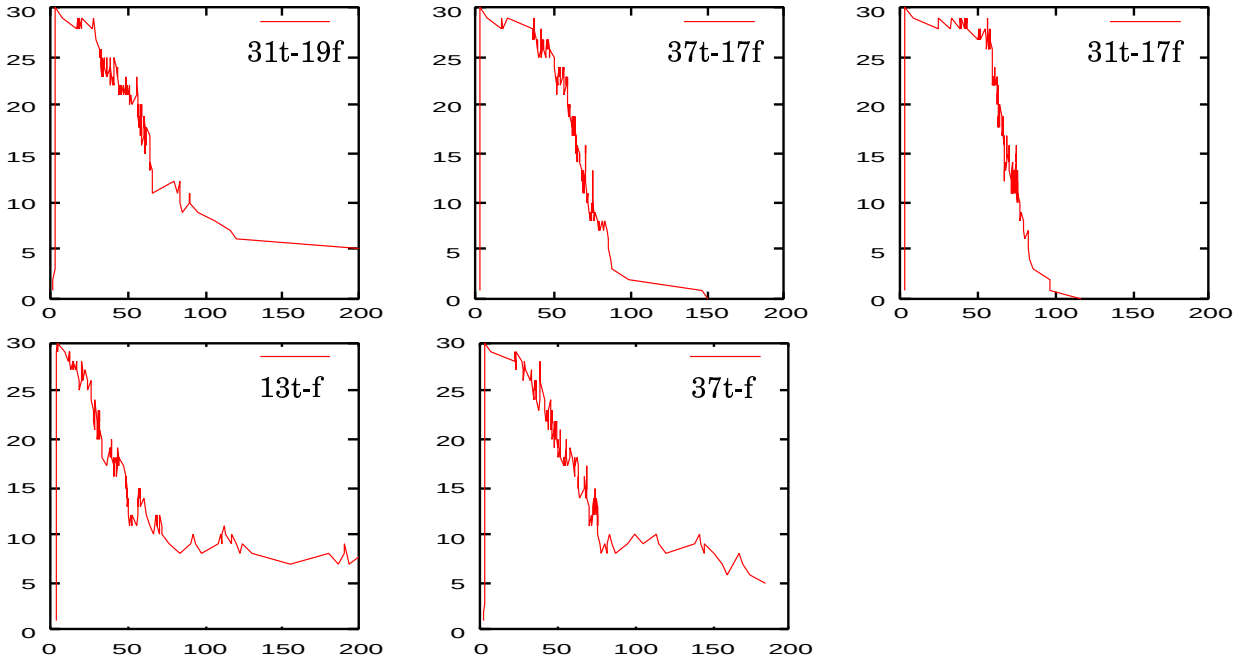

図 3: ハッシュ関数による稼働マシン数と時間の関係グラフ

くない結果になることもある. そこで考えられる手法として先読みのアルゴリズムを提案する. 前述したように並列度の向上のために後半で生成されるであろう状態式を前半で予測を行う.

時相論理式を展開していくとある共通の部分項またはそれを表す BDD が出て来る. そこで BDD のノードを調べ, 他の状態式の部分項となっている頻度の高い BDD を計算に使われていない計算機に展開させることによって, 状態式の展開の負荷分散を上げることができるかもしれない.

例えばある式の部分 BDD が以下のように表される状態式を考える。

?(3,?(12,?(35,T,F),?(49,?(12,?(35,T,F),F),
?(35,T,F))),?(5,T,F)).

これをツリー状に直すと図 5 となる. 図 5 より, この BDD 上では 35 のノード番号の BDD が他の BDD のノード番号よりも多く出現していることが分かる. ここで, この 35 のノード番号に一致する状態式は全ての分散処理系で一様でなければならないので, この部分項の展開を他の部分項のノードよりも先読みこんで検証を行うことで, 全体の式にかかる手間が落ちる可能性もある. 以下のような例も挙げられる.

例 1 ：

ITL において時空間の結合を表すチョップオペレータ (&) は容易に次に行うべき式を探すことが可能である. $P\&Q$ の式の先読みを行う場合は右辺の式 Q の先読みが可能である.

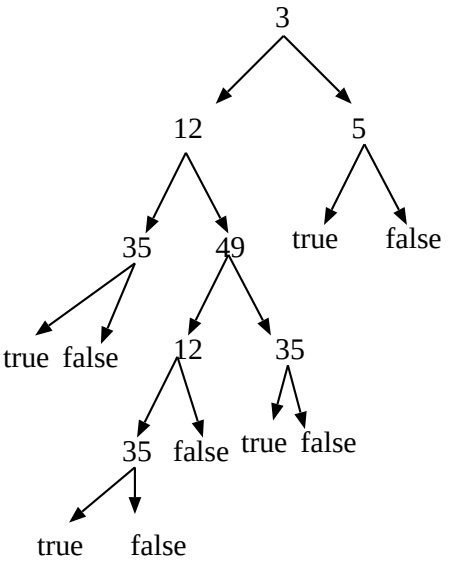

図 5: BDD の例

例 2 ：

Next オペレータ ◯ も同様である. ◯◯...◯P の場合、◯P の先読みが可能である. 最右辺の

◯P の先読みを行い, 検証結果を得ることができる.

これらの ITL の論理をうまく利用することで, 展開式を別々の計算機上で待つこと無く展開を行うことができ, でてきた展開結果の組合せを作れば, 検証も効率よくなると考える.

7 まとめ

今回はいくつかのハッシュ関数を用いて負荷分散について評価を行った. ハッシュ関数はその定数により負荷分散の状態も違うものになっている. しかし, それが全ての論理式を同様な形に分配を行うとは限らない. この時グラフの長く延びている部分の処理を, 他のノードに割り当てることにより並列度をあげることができる.

対処方法としては以下に述べる先読みアルゴリズムの提案が挙げられる.

これまで述べた工夫は, 全体の計算速度を上げるものだが, 検証できる仕様の大きさを制限するのは状態式の集合の大きさそのものであり, それ自体を小さくする必要がある.

対称性の利用や, 青島 [5] などによる状態の分割によるタブローの証明系の効率化の手法があるが, 仕様記述の基本単位を変更し, より検証しやすい, 状態式の集合が小さくなるような仕様記述の方法を研究することが重要だと思われる. それは, 実際には, 仕様記述に対するある種の制限になる. 例えば, 仕様記述の単一代入化, あるいは, 決定的な場合分けなどの制限が有効であると期待される.

7.1 今後の課題

今後の課題として先読みアルゴリズムの導入を行い, 並列タブロー法の効率を向上させる. また, このアルゴリズムを導入する上でのクラスが見付かれば効率の良い検証法ができると思われる.

また, 本稿では Prolog を用いて検証を行った. 次の目標として,Java を用いた検証アプリケーションの作成も考えている.

参考文献

[1] Masahiro Fujita, Shinji Kono, Hidehiko Tanaka, Tatsuhiko Moto-oka, Tokio: Logic Programming Language Based on Temporal Logic and Its compilation to Prolog, Internatinal Conference on Logic Programming, Sept. 1986

[2] Masahiro Fujita and Shinji Kono, Synthesis of Controllers from Interval Temporal Logic Specification, In *International Conference on Computer Design*, Masssachusetts, Oct. 1993

[3] 河野 真治, 池村 正之, 状態集合の分割による時相論理検証の並列化, 電気学会・電気通信学会合同講演会, Dec. 1998

[4] 河野 真治, 神里 健司, UDP を使った分散計算環境とその応用, 日本ソフトウェア科学会 第 16 回大会論文集, Sept. 1999

[5] 青島武伸、米崎直樹, 部分評価を用いる時相論理タブロー証明系の効率化, 日本ソフトウェア科学会 第 17 回大会論文集 Sept. 2000