正則な項書換え系のマルチプロセッサ上での一実現法

4B－1

岡野 浩三　松石 航也　東野 輝夫　谷口 健一

大阪大学 大学院基礎工学研究科 情報数理系専攻

1　まえがき

本稿では，正則な項書換え系 (TRS) を対象とし，TRS の並列性を利用した効率の良い (基礎項の) 書換え方法とその評価を報告する．並列計算機のモデルとして，局所メモリを持つ複数のプロセッサと共有メモリからなる非同期式並列計算機モデルを採用する．正則な TRS においては，最外リデックスが有限ステップ内に書換えられることが保証されるならば，正規形が (もしあれば) 求められることが知られている [3]．本アルゴリズムはこの戦略に基づき，次の工夫を行なった．

マッチングの際に，有限オートマトンを用いる [1, 5, 2]．本研究では [5] の 1 プロセッサ上ですべての最外リデックスを見つけるアルゴリズムを並列計算機上で動作するように改良した．一般に各部分項における書換え時間は均一でないため，並列計算機上で項書換えを効率良く行なうにはマッチングや書換えのタイミングを調整して，書換え最中の部分項とマッチするなどの誤った変形操作を防ぐ (項の一貫性を保つ) 必要がある．そこで，過去のマッチングオートマトンの使用情報とプロセスの生起関係を利用することにより 対応した．

本実現法の評価を行うために，シミュレータを逐次型計算機上で作成した．提案する方法と単純ないくつかの方法に対し，並列度と速度についてそれらの方法を比較した．

2　並列項書き換え

書換え対象となる項 (対象項) は木表現で共有メモリに置く．木の各ノードは 対応する関数記号，子ノードに対するポインタ等の情報を持つ．

最初は，1 つのプロセッサが 対象項に対するマッチング (どの規則とマッチするかを調べること) を実行する．処理の進行に伴い，最外リデックスの複数の候補 (部分項) が見つかる．これらの各候補に対して，空いているプロセッサが割当てられる．個々の候補に対して行なうべき処理を “プロセス” と呼ぶ．プロセッサが有限であるので，プロセスはプロセスキューで管理する．

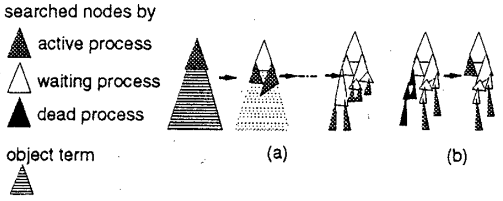


図 1: 書換え過程

[基本方針]

1. より一般には，プロセスキューから取り出されたプロセス p は，割当てられた候補 (部分項 $ST(p)$) に対するマッチングを実行する．マッチングには後述のマッチングオートマトン (MA) を用いる．ある規則とマッチングするのであれば p がその規則で $ST(p)$ の書換えを行なう．
2. もしマッチングする規則が全くなければ，MA の結果から，$ST(p)$ の複数の部分項が最外リデックスの候補として得られる．p はその候補の分だけ子プロセスを複数生成し，各子プロセスがそれぞれの候補を担当する (図 1(a))．プロセス p は全ての子孫プロセスが死ぬまで休止する．
3. もし次候補がなければ $ST(p)$ は既約であり，この先書換えを行なう必要はない．そこで $ST(p)$ に既約のマークをつけ p は死ぬ．
4. 一般にある項 t の部分項の書換えが生じると，t が新たに最外リデックスになる可能がある．ここでは，項の一貫性を保証するため，t のすべての部分項の書換えが終了してから，t のマッチングを再実行する (休止していたプロセスが再び活動状態になり，マッチングを再実行する (図 1(b)))．一般に，幾つかの部分項の書換えがすべて終了した時点で，その影響を受け，リデックスになる可能の項は複数生じる．ここでは簡単のため，そのような項のうち，最上位の項のみマッチングを再実行 (プロセスの再起動) し，それ以外の項のマッチングは行なわない (該当プロセスは死ぬ)．
5. プロセスキューのプロセスは，アイドルプロセッサに先頭から割当てられる．

[基本指針] の 4 に関して，全ての部分項の書換えの終了を待たず上位の項のマッチングや書換えを再開する方法も考えられる．この際，マッチング中あるいは書換え中の部分項にまたがる マッチングや書換えが生じるため，プロセス間で優先順に関する合意が必要であり，メッセージの増大が予想される．本稿では,4 の方法をとり，メッセージ交換を先祖子

An Implementation of Orthogonal TRS on Multi-Processor Machine

Kozo OKANO, Koya MATSUISHI, Teruo HIGASHINO and Kenichi TANIGUCHI

Division of Infnformatics and Mathematical Science, Osaka University, Machikaneyama 1-3, Toyonaka, Osaka 560, JAPAN

孫プロセス間 (対象項の項と部分項の関係にも対応する) に限ることとした.

[基本指針] の 4 のためには, (I) 「項の下位ノードの書換えにより, 項の上位ノードでマッチする可能性が生じた場合, どの上位のノードまで影響を受けるのか? また, 書換え終了をどのように知らせるのか?」, (II) 「影響を受けるノード集合のうち最上位のノードをどのように選ぶのか?」という問題と, (III) 「再実行時にどのようにマッチングを再開すれば効率がよいか?」という問題を解決する必要がある. (III) に関しては, MA の失敗時の情報を保持し, 再利用することにより解決する. 一方 (I),(II) に関しては, 「基本方針ではプロセスの親子関係が, 担当する部分項の位置関係と対応していること」を利用し, プロセス p による書換えによって影響を受ける可能性のある先祖プロセス の集合 $F(p)$ を求める. $F(p)$ が求まれば, 書換え終了のメッセージの伝達範囲や, 最先祖プロセスを求めるのは容易である.

2.1 マッチング履歴情報の利用

MA は検査対象の部分項の関数記号を入力とする. 関数記号は幅優先探索順序で根ノードの関数記号から入力され, 最終状態はマッチングする規則の有無を与える. また, マッチする場合はマッチする規則が何であるかを与える. マッチングする規則がない場合は, 次のマッチング時に開始すべきオートマトンの状態と入力部分項の適切な位置 (失敗情報) を知ることができる (上述の (III)).

(I),(II) に関しては次のマッチング履歴を用いる.

マッチング履歴は以下の項目 (プロセス情報) を現在の全ての休止, 活動プロセス p に対して (プロセスの親子にそって) 集めたものである.

失敗情報, 親プロセスの位置するノードから p が位置するノードまでの距離, 子プロセスの数 等.

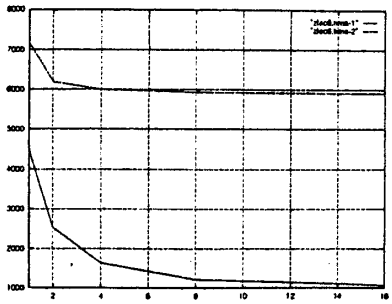
$F(p)$ 中の各プロセッサは書換えのあったノードから高々規則左辺項の "深さ" まで対象項を遡ったノードに位置している. これとマッチング履歴を用いて $F(p)$ が求められる. また, $F(p)$ に属する各休止プロセスは, 順次, すべての子孫プロセスの死の情報を得て, (自プロセスより先祖のプロセスの情報に頼らなくても,) 再びマッチングを再実行すべきか, それとも直ちに死ぬべきか等を知ることができる. また, マッチングを再実行する際に失敗情報を用いることにより, より適切な状態からマッチングを再実行することができる.

3 評価

C++用の並列プログラムとシミュレーション用ライブラリ AWESIME [4] を用いて, 次の 2 つの方針の実現法をシミュレートするシステムをそれぞれ作成した.

アルゴリズム 1 本実現法による書換え方法

アルゴリズム 2 履歴情報を用いず, 対象とする項の根からのマッチングを繰り返し, 毎回すべての最外リデックスを 1 回ずつ書換える方法



6!の計算

実線:アルゴリズム 1 破線:アルゴリズム 2

横軸はプロセッサ台数 (2 台単位)

図 2: 実行結果

各方法について, プロセッサ数が 1,2,4,8,16 台の計算機で実行したときの時間 (ノードの作成, 参照などの操作を 1step とした) をシミュレーションでそれぞれの方法について計測した. 例題として階乗, フィボナッチ数列, n クイーン問題, クイックソートを用いた. 図 2 に 階乗プログラムの実験結果を表す.

階乗やフィボナッチ数を求める問題では, 非常に大きな差が出ている. これは, 本実現法でのマッチングの履歴を利用する効果によるものと思われる. また, プロセッサ数の増加効果も, 大きいことが分かった. 特に, n クイーン問題ではその差が顕著であり, これは規則の個数が多いためと思われる.

4 あとがき

正則な TRS に対し, 有限個のプロセッサからなる共有メモリ型並列計算機上で, 効率良く正規形を求めるための方法を提案し, その評価を行なった.

本実現法は, 扱う項が大きい場合や規則の個数が多い場合には, マッチング履歴がうまく働くことにより, 並列度が高く高速に正規形を得ることができることが分かった.

参考文献

[1] Burghardt, J. : "A Tree Pattern Matching Algorithm with Reasonable Space Requirements", Lecture Notes in Computer Science 299, pp.1-15 (1988).

[2] Hoffmann, C. M. and O'donnell, M. J. : "Pattern Matching in Trees", Journal of ACM, Vol.29, No. 1 (1982).

[3] O'donnell, M. J. : "Computing in Systems Described by Equations", Lecture Notes in Computer Science 58, (1977).

[4] Grunwald, D. : "A Users Guide to AWESIME : An Object Oriented Parallel Programming and Simulation System", Univ. of Colorado Technical Report CU-CS-552-91, (1991).

[5] 山本, 坂部, 稲垣: " 項書換え系における並列最外戦略の効率的な実現法", 信学論, J77-D-I, No.10, pp.693-702, (1994).