並列一般化LRパーザの負荷分散の検討

沼崎 浩明 田中 穂積
Hiroaki NUMAZAKI Hozumi TANAKA

東京工業大学 工学部
Tokyo Institute of Technology
〒152 目黒区大岡山 2-12-1
03-726-1111 (内線 4175)
E-mail : numazaki@cs.titech.ac.jp

概要

我々は，LR構文解析法[Knuth 65]に基づく並列パーザを並列論理型言語KL1で実現した．この並列LRパーザは文の構文的解釈の曖昧性を並列に計算する．ただし，複数のプロセスの解析過程が重複する場合は，これを一つのプロセスに統合する．我々の方法はこの点で富田法[Tomita 85]に基づいている．本報告では，我々の並列LRパーザをマルチPSI実機で実行するための負荷分散の方法を検討する．そこでは実際に，CFGで記述した英語の文法規則を用いて実験を行ない，解析時間の台数効果を測定した．

1 まえがき

本報告では，KL1で記述した並列LRパーザ[沼崎 89]をマルチPSI実機で実行するための負荷分散の方法について検討する．LR構文解析法を一般の文脈自由文法に適用すると，パーザを駆動するLRパーズ表にコンフリクトを生じ，そこで解析が非決定的となる．我々の並列LRパーザは，コンフリクトの生じたエントリが指定する複数の処理を，それぞれ異なったプロセッサ上で実行する．これにより，自然言語の構文的な曖昧性が並列に処理される．

本報告では並列一般化LRパーザのKL1による実現方法を簡単に示し，これをマルチPSI実機で実行するための負荷分散の方法を示す．マルチPSIのような疎結合型の並列計算機では，プロセッサ間の通信のオーバヘッドが問題となる．我々のLRパーザは複数のプロセスの解析が重複する場合，これを一つに統合する．そのためには，一方のプロセッサの持つスタックの情報をもう一方のプロセッサに転送する必要がある．各スタックには，それが得られるまでの解析の情報が全て含まれているため，通信のオーバヘッドは解析の効率の点から無視できない大きさとなる．そこで我々は，プロセッサ間の通信量を抑えるために，一旦負荷分散したプロセスは，他のプロセッサ上にあるプロセスとは通信しないようなインプリメントを行なった．そこでは，異なるプロセッサ上のプロセスの重複を許す．しかし，一つのプロセッサが複数のパージングプロセスを扱う必要が生じた場合は，スタックの木構造化によって重複した解析を防ぐ．

(1) S → NP, VP.
(2) S → S, PP.
(3) NP → NP, PP.
(4) NP → det, noun.
(5) NP → pron.
(6) VP → v, NP.
(7) PP → p, NP.

図 1: 曖昧な英語の文法

	det	noun	pron	v	p	$	NP	PP	VP	S
0	sh1		sh2				4			3
1		sh5								
2				re5	re5	re5				
3					sh6	acc		7		
4				sh8	sh6			10	9	
5				re4	re4	re4				
6	sh1		sh2				11			
7					re2	re2				
8	sh1		sh2				12			
9					re1	re1				
10				re3	re3	re3				
11				re7	sh6/re7	re7		10		
12					sh6/re6	re6		10		

図 2: LR パーズ表

本方式をマルチ PSI 実機で実行し，その場合の台数効果を測定するために，規則数 123 の文脈自由文法を用いて，使用するプロセッサの台数に対する解析時間の推移を測定した．その結果，最良値として，2 倍程度の台数効果が得られた．この値は寿崎らの報告 [寿崎 89] を参考にすれば，初期の実験結果としては妥当な値であると言える．

今回実験した負荷分散の方法は，一台のプロセッサに一回しかプロセスを投げないため，そのプロセスの実行が終了すると，そのプロセッサは稼働しなくなる．この点は今後改善する必要がある．

2 KL1 による並列一般化 LR パーザの記述

本節では，並列 LR パーザの KL1 による記述を示す．これを説明するために簡単な英語の CFG 文法と，それから得られる LR パーズ表を図 1, 図 2に示す．図 2の p の列の 11,12 の行にはコンフリクトが生じている．このようなエントリをシフト・レデュースコンフリクトと呼ぶ．パージングプロセスはこのエントリに到達すると，二つに分かれてそれぞれの解析を行なう．

並列一般化 LR パーザの KL1 による実現の特徴は，LR パーズ表の各エントリをパージングプロセスの記述に置き換え，そこでスタックを操作する点にある．パージングプロセスは以下の 5 つのプロセスから構成される．

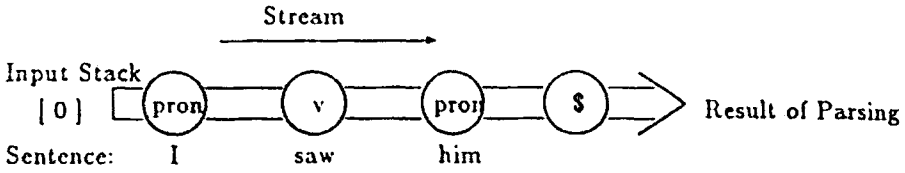


図 3: 親プロセスとストリーム

1. 親プロセス
   入力文中の各入力語に対して起動するプロセス．このプロセス間にストリームを張り，そこにスタックを流す．各親プロセスは入力ストリームにスタックを一つ受けとると，エントリプロセスを呼び出してそのスタックを処理する．その結果得られたスタックは直ちに出力ストリームに送られ，それが次の親プロセスの入力となる (図3参照)．親プロセスは文法規則中の任意の前終端記号 (preterminal) に対して記述する．

2. エントリプロセス
   LR パーズ表の各エントリのスタックの操作を実行するプロセス．このプロセスは，親プロセスから先読み語のカテゴリと，スタックの先頭の状態番号によって定められる．各エントリプロセスに対して一つずつスタックが渡される．

3. reduce プロセス
   スタックを必要な数だけ reduce するプロセス．reduce が木構造化スタックの分岐点に及ぶと，その先のそれぞれの枝に対して別々の reduce 操作を実行する．

4. condition プロセス
   規則が DCG 文法で与えられている場合，reduce 操作の際にその補強項を実行するプロセス．補強項の成功／失敗によってパージングプロセスの継続／終了を判定する．

5. merge_stack プロセス
   複数のスタックの先頭の要素が等しい場合，その部分を統合して一つの木構造化スタックを生成するプロセス．以下に木構造化したスタックの例を示す．

```
[ 6,p, [12,np,8,v,4,np,0], [3,s,0] ]
```

- 親プロセスの記述例
  例えば，カテゴリ p に対する親プロセスは，

```
p( [], _, Out ):- true |
   Out = [].
p( [Stack|Stream], Cat, Out ):- true |
   p_( Stack, Cat, Out1 ),
   p( Stream, Cat, Out2 ),
   merge_stack( Out1, Out2, Out ).
```

と記述する．ここで，P_はエントリプロセスであり，各エントリに対して以下のように記述する．

- シフトエントリの記述例
状態1，先読み語がnounのエントリ'sh 5'は，

```
noun_( [1|Stack], Cat, Out ):- true |
    Out = [ [ 5,Cat,1 | Stack ] ].
```

と記述する．

- レデュースエントリの記述例
状態2，先読み語が'v'のエントリ're 5'は，

```
v_( [2,T1|Stack], Cat, Out ):- true |
    cond( 5, Stack, [T], Result ),
    v(Result, Cat, Out ).
```

と記述する．

- シフト・レデュースコンフリクトのあるエントリの記述例
状態11，先読語がpのエントリ'sh 6/re 7'は，

```
p_( [11,T|Stack], Cat, Out ):- true |
    reduce( 7, 1, Stack, [T], Result ),
    p( Result, Cat, Out1 ),
    merge_stack( [ [6,Cat,11,T|Stack] ], Out1, Out ).
```

と記述する．reduceプロセスは第二引数に与えられた数だけスタックの要素をポップし，それをconditionプロセスに渡す．

- パーザの起動
例えば，入力文の各単語のカテゴリがpron，v，nで，それに付随する情報がPron，V，Nの時，次のような親プロセスの列を呼ぶことによってパーザを起動する．

```
?- pron( [ [0] ], Pron, Stream1 ),
    v( Stream1, V, Stream2 ),
    n( Stream2, N, Stream3 ),
    $( Stream3, [], Result ).
```

最初のゴールpronに一つのスタック［0］を与える．

3 負荷分散

この節では，並列LRパーザの負荷分散について図4を用いて説明する．図4(a)(b)(c)の各円はパージングプロセスのある時点での状態を示し，プロセスは左から右へと推移していくものとする．最初は一つのプロセスで入力文を解析していき，解析が曖昧になると複数のプロセスに分かれる．(図(a)参照)．それぞれのプロセスには重複する部分が存在するため，これを削減すると全体の計算量が減る(図(b)参照)．しかし，マルチPSIのような疎結合型の計算機では，各プロセスをそのまま負荷分散すると通信コストが膨大になる．なぜなら，図(b)の全ての斜め線の部分でプロセスの間の通信が起こるためである．

そこで本報告では，異なるプロセッサ上に移動したプロセスは，他のプロセッサ上にあるプロセスとは通信しないような負荷分散を実現した(図(c)参照)．プロセッサ数の制限から一つのプロセッサが複数のプロセスを扱う場合は，スタックの木構造化によって解析の重複を避ける．図(c)は使用するプロセッサの台数が4台の場合のプロセスの推移を示している．この図中の3箇所の矢印で示された部分でプロセッサ間の通信が起こる．

負荷分散は以下のようにして行なう．

- 各プロセスは，自分がこれから負荷分散を行なうことができるプロセッサの台数を持つ．
- 負荷分散を行なう際に，プロセス数が使用可能なプロセッサ数よりも多い場合は，各プロセッサにできるだけ均等にプロセスを割り当てる．この場合，分散された各プロセスはそれ以上負荷分散を行なわない．
- 負荷分散を行なう際に，プロセス数が使用可能なプロセッサ数よりも少ない場合は，使用可能なプロセッサ数をできるだけ均等に分割して，各プロセスに割り当てる．

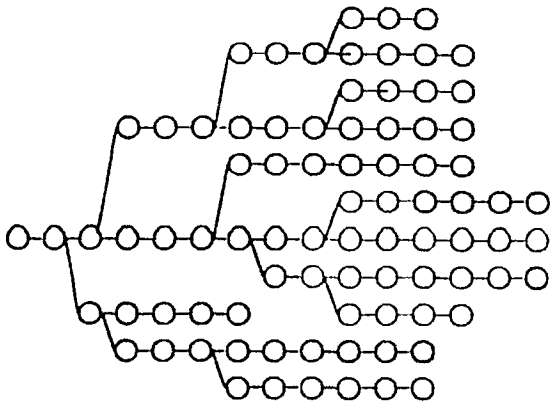
図(c)中の数字はプロセスに割り当てられたプロセッサの台数を示している．ここでは，最初に使用可能なプロセッサの台数は4台であり，プロセスはPE0で実行される．そのプロセスが二つに分岐する時，一方のプロセスはPE0で実行を継続し，もう一方のプロセスはPE2へ移動する．その際，各プロセスに割り当てられるプロセッサ数は2台となる．この図のように，異なるプロセッサ上のプロセス間では，通信を行なわないようにインプリメントした．これによって，通信のコストがある程度大きくても，各プロセッサ上の処理量がそれ以上に大きければ，台数効果を期待できる．ただし，この負荷分散の方法は次の二つの問題点がある．

- あるプロセッサ上の全てのプロセスが処理を終えると，そのプロセッサは二度と稼働しない．
- あるパージングプロセスが，自分に割り当てられたプロセッサの全てを使い尽くさない場合，最初から全く稼働しないプロセッサが生ずる．

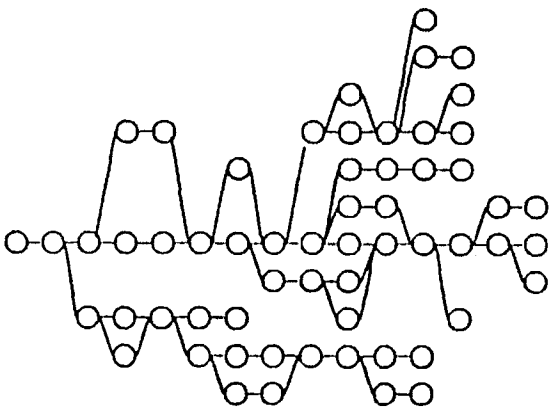
この二点は今後，動的負荷分散などの方法を導入することによって改善する必要がある．

4 実験

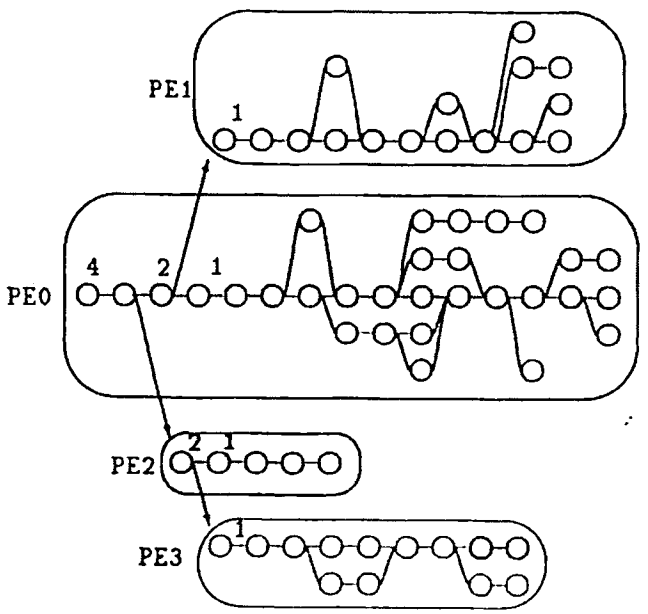
今回の実験では，規則数123の文脈自由文法を用いて，本方式を実現する並列LRパーザを構築し，以下の5つの入力文に対する解析時間を，使用するプロセッサの台数を変えて測定した．使用した計算機は16台のプロセッサを有するマルチPSI実機である．



(a) プロセスを統合しない場合



(b) 統合可能なプロセスを全て統合した場合



(c) 同一プロセッサ上のプロセスのみ統合する本方式
(プロセッサ数 4 台の場合の例)

図 4: プロセスの推移

文	木の数	プロセッサの台数	Reduction 数	解析時間(ms)
1	30	1	7550	671
		2	7776	522
		4	8306	483
		6	8358	416
2	56	1	14605	1521
		2	15140	1196
		4	15213	1018
		8	16739	938
		10	17543	854
		12	17535	913
3	192	1	88291	7765
		2	92504	6302
		4	92409	6241
		8	102335	5457
		10	111435	5223
		12	113222	5960
4	200	1	142567	11392
		2	135209	6005
		4	175675	7102
		8	199789	7832
		12	203661	7007
5	186	1	16972	1388
		2	16981	1331
		4	20492	1250
		8	23169	1230
		12	24445	943
		16	24506	885

表 1: 解析時間の推移

1. Diagram analyzes all of the basic kinds of phrases and sentences.

2. This paper presents an explanatory overview of a large and complex grammar that is used in a sentence.

3. The annotations provide important information for other parts of the system that interpret the expression in the context of a dialogue.

4. For every expression it analyzes, diagram provides an annotated description of the structural relations holding among its constituents.

5. Procedures can also assign scores to an analysis, rating some applications of a rule as probable or as unlikely.

表 1に解析時間と reduction 数を示す.

5 おわりに

今回の実験では最良で2倍程度の台数効果による解析効率の向上を得ることができた. 寿崎らの報告 [寿崎 89] を参考にすれば, この値は初期の実験として妥当であると思われる.

今後，負荷分散の方式の改善や通信する情報の圧縮などを行なえば，さらに良い結果が得られると期待できる．

謝辞
本研究を進めるにあたり，日頃から暖かいご支援をいただいた田中研究室のみなさんに感謝致します．また，本研究に対し貴重な御意見をいただいたICOTのKL1タスクグループの方々に感謝致します．

参考文献

[寿崎 89] 寿崎かすみ，他:マルチPSIにおける並列構文解析プログラムPAXの実現および評価，並列処理シンポジウムJSPP '89, PP.343-350 (1989)

[沼崎 89] 沼崎浩明，田村直良，田中穂積:並列論理型言語による一般化LR構文解析アルゴリズムの実現，自然言語処理 74-5, PP.33-40 (1989)

[沼崎 90] 沼崎浩明，田中穂積:LR法に基づく新しい並列構文解析アルゴリズム，情報処理学会第40回全国大会, 4F-3, pp.456-457 (1990)

[峯 89] 峯　恒憲，谷口倫一郎，雨宮真人:文脈自由文法の並列構文解析，情報処理学会自然言語処理研究会研究報告，73-1, pp.1-8 (1989)

[Aho 72] Aho,A.V.and Ulman,J.D.: *The Theory of Parsing,Translation,and Compiling,* Prentice-Hall,Englewood Cliffs,New Jersey (1972)

[Henry 89] Henry S. Thompson:*Chart Parsing for Loosely Coupled Parallel Systems,* Proc. of International Workshop on Parsing Technologies pp.320-328 (1989)

[Knuth 65] Knuth,D.E.: *On the translation of languages from left to right,*Information and Control 8:6,pp.607-639

[Matsumoto 88] Matsumoto,Y.:*Natural Language Parsing System based on Logic Programming,* Doctoral Thesis, Kyoto University (1988)

[Nijholt 89] Anton Nijholt:*Parallel Parsing Strategies in Natural Language Processing,* Proc. of International Workshop on Parsing Technologies pp.240-253 (1989)

[Tomita 85] Tomita,M.:*Efficient Parsing for Natural Language,* Kluwer Academic Publishers (1985)

[Tomita 87] Tomita,M.: *An Efficien Augmented-Context-Free Parsing Algorithm,* Computational Linguistics, Vol.13, Numbers 1-2, pp.31-46 (1987)

[Ueda 85] Ueda,K:*Guarded Horn Clauses,* Proc. The Logic Programming Conference, Lecture Notes in Computer Science, 221 (1985)