並列構文解析システム PAX の OR 並列化による改良

津邑公暁[†] 五島正裕[†] 森 眞一郎[†]
中島 浩[†] 富田眞治[†]

本稿では，並列構文解析システム PAX の高速化手法を示す．PAX では AND 並列性に基づく並列処理方式がとられているため，単語数以上の並列性は得られない．そこで本方式では OR 並列性に基づく負荷分散方式を PAX に採用する．OR 並列性とは構文解析における曖昧性に対応する．従って，単語数に縛られない並列性が引き出せる上，プロセッサ間のデータ依存性も軽減する．

本方式を AP1000 版 KLIC 上で評価した結果，文法規則約 500，単語数約 200 からなる構文において，単語数 20 からなる文を 20 台のプロセッサを用いて処理した場合，ゴール中断回数，メッセージ数ともに約 1/4 に減少した．

An Improvement of a Parallel Parsing System PAX by OR-parallelizing

Tomoaki Tsumura,[†] Masahiro Goshima,[†] Shin-ichiro Mori,[†]
Hiroshi Nakashima[†] and Shinji Tomita[†]

We improve a load-balancing method for PAX the parallel parsing system. Since PAX is based on the AND-parallelism, we cannot achieve the parallelism greater than the number of words in a sentence. Our method is based upon the OR-parallelism, which is correspond to the ambiguity in parsing. Hence, OR-parallelism is independent of the number of the words and decreases the inter-processor data dependency.

We evaluate the performance of this method on KLIC on AP1000, with a DCG which have about 500 syntax rules and about 200 words. Both the number of suspensions and the number of inter-processor communications become about 1/4 of PAX.

1. はじめに

自然言語処理は多大な計算量を必要とし，処理速度の向上が望まれる分野のひとつである．中でも比較的アルゴリズムの確立している構文解析の並列化が，近年活発に研究されてきた．新世代コンピュータ開発機構 (ICOT) で研究された並列構文解析システム PAX(Parallel Analizer for syntaX and semantiCS)[1)]もそのひとつである．

並列論理型言語 KL1[2)]で記述される PAX は，並列性を持つ上にバックトラックおよび副作用を伴わないという特徴がある．しかしこの PAX は通信量が多いという欠点があるため，それを低減するための改良方式が提案された[3)]．本稿ではこの改良方式を施した PAX を対象に議論を進める．

PAX のアルゴリズムは AND 並列性と OR 並列性を内包しているが，AND 並列性のみに基づいた並列処理を行っている．この方法では利用可能なプロセッサ数に関わらず，最大でも単語数分の並列性しか出ないうえ，プロセッサ間のデータ依存性も大きい．そこで今回，PAX に対して OR 並列性に基づく負荷分散を導入し，AP1000 版の KLIC 上でその評価を行った．また，更なる効率化のため，master-slave 方式による動的負荷分散を提案する．

以下，次章で背景として今回のターゲットである PAX とその記述言語 KL1 について簡単に触れる．第 3章で PAX の OR 並列性と改良案について述べ，第 4章で今回実装を行った OR 並列性的負荷分散方式とその評価結果を述べる．第 5章で負荷の均一化のための動的負荷分散方式の提案を行い，最後に第 6章で結論を述べる．

2. 背景

本章では，PAX の記述言語である並列論理型言語 KL1 の概要を説明し，PAX のアルゴリズムとその並列処理方式について概観する．

2.1 並列論理型言語 KL1

KL1 は，第五世代計算機計画の核言語として開発された並列論理型言語で，シンタクスが単純であること

[†] 京都大学大学院工学研究科情報工学教室
Department of Information Science, Faculty of Engineering, Kyoto University

⋮

vp1 → v, np.　　vp1 → v, np, adv.
vp2 → v, obj, adj.　　vp2 → v, obj.
vp2 → v, prep, obj.　　pred → v, obj.

⋮

図 1　DCG による簡単な構文規則の例 (部分)

や並列性を自然に記述できることなどの特徴を持つ．

2.1.1　プログラムの構造と実行

KL1 は Flat GHC に基づく言語で，以下の形をした節の集合で表される．

$$H\ :-\ G_1\ ,..\ ,G_m\ \mid\ B_1\ ,..\ ,B_n.$$

H, G, Bは，それぞれクローズヘッド，ガードゴール，ボディゴールと呼ばれる．

実行中ゴールのヘッドやガードで未具体化変数への参照が行われると，そのゴールは実行を中断 (suspend) し，別のゴールが選択されて処理が継続される．中断されたゴールは，その変数が具体化されると実行を再開する．すべてのノード上において実行可能なゴールがなくなると，プログラム全体の実行が終了する．

KL1 で負荷分散を行うには，明示的にプログラム中にこれを指定する．`goal(`$Args$`)@node(`N_C`)` という形のボディゴールがノード N_P上に現れると，このゴールはノード N_C上に送られ (throw goal)，そこで実行される．また，`@priority` プラグマによってゴールの実行優先度を指定することができる．

2.1.2　プロセスとストリーム

KL1 プログラムではボディゴールとして自身を再帰的に呼び出すことによって，ある条件の満たされる間存在し続けるようなプロセスを実現することができる．

KL1 プログラムではリスト変数を共有することによってプロセス間の通信路を実現できる．これをストリームと呼ぶ．また KL1 には組み込み述語として，入力ストリーム数が動的に変化する場合にも対応したマージャ`merger/2` が用意されている．

2.2　PAX のアルゴリズム

本節では PAX による解析法と，その並列処理方式について述べる．

2.2.1　PAX による解析法

並列構文解析法は構文解析において非決定性を扱う方法の一つで，互いに必ずしも両立しない複数の仮説を一時に保持する方法である．

並列構文解析法の一つにチャート法[4)]があるが，PAX は上昇型のチャート法を並列論理型言語で実現したものである．PAX はボトムアップを基本とし，これにトップダウン的予測を取り入れた左隅解析法[5)]と呼ばれる方法によって解析を行う．また，解析中に現れる曖昧性を保持するのに，前述のストリームを層状にして用いる[6),7)]．

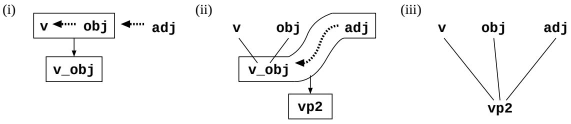


図 2　解析木が構成されてゆく様子

PAX は解析状況にプロセスを対応させ，それらの間で “あるまとまった終端記号や非終端記号” をデータとしてやり取りさせる．以下本稿では，これら非終端記号のことを言語学に倣いカテゴリと呼ぶことにする．

PAX のアルゴリズムは基本的にあるカテゴリが解析されると左のプロセスに自分がどういうカテゴリであるかを知らせる．図 1に示した DCG[8)] 内の vp2 → v, obj, adj. という文法規則に沿って解析木が完成していく様子を図 2に示す．この例では文法規則の右辺左隅のカテゴリに相当するプロセス `v` は，自分の右に `obj` というカテゴリを許すことを知っており (i)，そのカテゴリが送られてくると更にその右がどういうカテゴリであるかを調べることによって解析を進めていく (ii)．ここで `v_obj` は，構文規則 vp2 → v, obj, adj. の obj まで解析が終わった状態であることを示すプロセスである．このプロセスに対する入力が `adj` であった場合は文法規則が完成し，新たに `vp2` というカテゴリに対応するプロセスが生成されて処理が進められてゆく (iii)．また，現在の解析状況と整合性のとれないものがデータとして送られてきた場合そのデータは破棄される．こうすることで解の候補が絞られてゆく．

2.2.2　PAX の並列処理方式

PAX の並列処理方式は，以下の通りである．まず単語間をマージャで結び，そのマージャにプロセッサを割り付ける．また単語に相当するプロセスを，そのプロセスへの入力マージャと同じプロセッサ上で起動する．これらプロセッサ上に割り当てられた各プロセスは，自らを右辺左隅にもつような構文規則を完成させようと，それぞれ並列に処理を行う．以降各プロセスは，マージャを通してデータを読み込もうとする際にはそのマージャのあるプロセッサに移動してからデータを読み込む．

これは入力マージャからデータを読んで出力マージャにデータを送る働きをするプロセスが多い PAX において，多対多の通信が起きるのを回避するために考え出された方法である．PAX ではストリームを流れるデータが非常に大きいため，データの移動よりもプロセスの移動の方がコストが小さい場合が多い．この方法を用いることによって同じマージャの出力を読もうとするプロセスはすべて 1 つのプロセッサに集まるため，通信は多対 1 となる．

図 1に示した構文規則に基づく解析の例を図 3に示す．ここで PE$_i$ 上のプロセス `v` はその入力として np を受け取ると子プロセス `v_np`, `vp1` を起動するが，これら子プロセスはその入力につながるマージャの存在

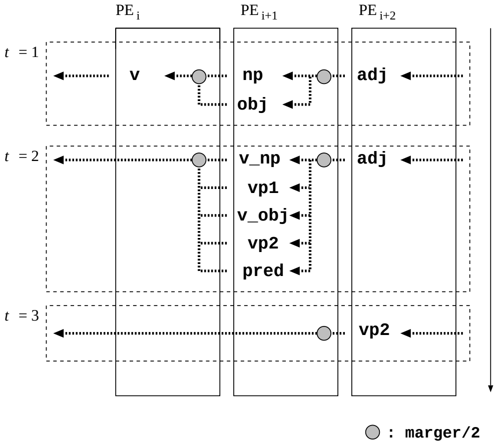


図 3　PAX による解析例

するプロセッサ PE$_{i+1}$ に移動する．入力 obj に対しても同様に，起動された子プロセス v_obj, vp2, pred は入力を読むために PE$_{i+1}$ に移動する．

3. 改良方式

PAX に OR 並列に基づく負荷分散方式を適用するにあたり，本章では今回の改良の方針を述べる．まず PAX の OR 並列性が 2 層から成っていることを説明したのち，負荷分散への OR 並列性の利用について考える．

3.1 方針

PAX のアルゴリズムには以下の AND/OR の 2 種類の並列性がある．それぞれについて以下に述べる．

AND 並列性

これは，文の各部でそれぞれ非終端記号を独立に生成することのできる並列性である．これらの間には，文全体に対応する解析木が構成されていく過程で整合がとられる必要がある．PAX の並列処理方式はあくまで AND 並列性のみを利用したものであるため，最大でも単語数分の並列性しかなく，プロセッサが多く利用できる環境であってもそれらを活かしきれない．また，ゴール間のデータ依存性が高くなり，ゴールの中断も頻繁に起こる可能性が高い．

OR 並列性

構文解析においては，解析対象である文には一般に曖昧性が含まれる．ある程度の大きさをもつ構文規則を用いて解析を行う場合，解が唯一であるときでも解析の過程においてはほとんどの場合そこに曖昧性が存在する．この解析過程における曖昧性は部分解の独立な候補が複数存在することに相当し，それら部分解は互いに OR 関係にある．これら部分解の間に存在する並列性が OR 並列性である．

今回は，AND 並列性に相当する部分を 1 プロセッサ内で行い，OR 並列性に相当する独立な探索処理を並列に行う方式の実装を行った．この方法では KL1 ゴールの中断回数やノード間にわたるメッセージの数の低減，延いては実行効率の向上が期待できる．

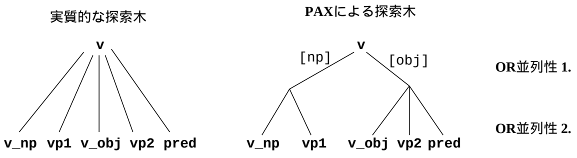


図 4　探索木に見る 2 層性

以下で，この OR 並列性に基づく PAX の並列化手法を述べるが，その前に並列化手法を考える際に問題となる点である，PAX の OR 並列のもつ 2 層性についてまとめておく．

3.2 OR 並列の 2 層性

3.1節で述べた OR 並列性を更に詳しく見ると，以下のように細分化できることがわかる．本稿ではこれを PAX の OR 並列がもつ 2 層性と呼ぶ．

OR 並列性 1.　あるプロセスがストリームを通して受け取る部分解析木の候補が複数ある場合に，それらの候補間に存在する並列性．

OR 並列性 2.　ある部分解析木の候補を受け取ったとき新しい部分解析木の生成に複数の可能性がある場合，それらの可能性間に存在する並列性．

つまり PAX において OR 並列に対応する処理は，複数の部分解の送受信と複数プロセスの起動という 2 つの段階から構成されている．図 3の例で $t = 1$ から $t = 2$ へと処理が進む際の探索木でこれを示すと図 4 のようになる．入力として np, obj を受け取った場合，{v_np, vp1} と {v_obj, vp2, pred} は共に v の子プロセスであるがそれぞれ別々に起動されることになる．この時，各入力 np, obj に対応するそれぞれの処理の間の関係が OR 並列性 1. であり，起動される子プロセス v_np, vp1 間，あるいは v_obj, vp2, pred 間の関係が OR 並列性 2. である．

3.3 OR 並列性の負荷分散への利用

以下に前述した 2 つの OR 並列性の特徴を述べ，負荷分散への利用可能性について考える．

OR 並列性 1. ではプロセスに対する入力の個数分の並列性が得られる．しかしプロセスへの入力は動的に決定されるため予めその全体を知ることはできず，どれだけの並列性が得られるかは実行するまで予測不可能である．例えば図 1の v に対応するプロセスに対し，図 3と同様入力ストリームに np, obj という 2 つのデータ (前段階までにおける部分解の候補) が送られてくる場合を考える．結果的にはプロセス v は 5 つのプロセス v_np, vp1, v_adj, vp2, pred を新たに起動することになるが，1 つめの候補である np を受け取った時点では，とりあえず 2 つの新たなプロセス v_np, vp1 を起動するということしか分からない (図 4 右)．よって OR 並列性 1. を静的な負荷分散に利用す

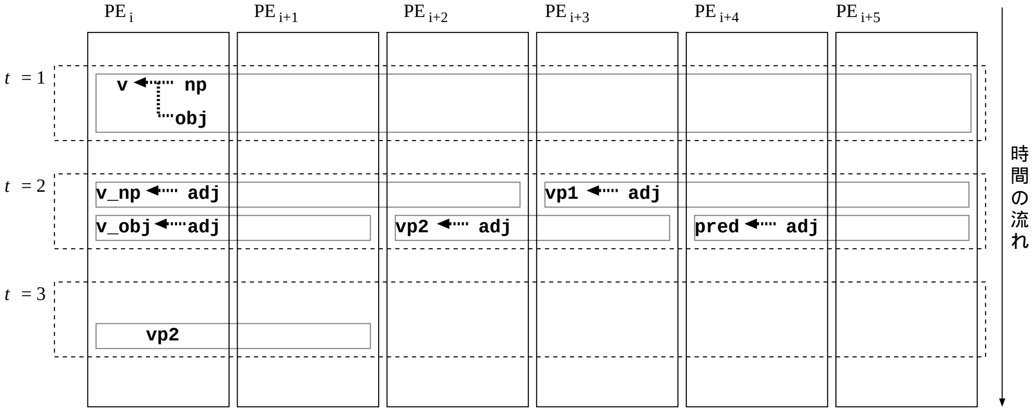


図 5　OR 並列に基づく負荷分散の例

ることは不可能である．

OR 並列性 2. の並列性はプロセスがある入力を受け取ったときに起動すべきプロセスの数であり，これは DCG から静的に決定可能である．KL1 のプログラム上では起動する子プロセスの数に対応する．但し OR 並列性 2. はその時点までの部分解である入力をプロセス間で共有することになるので，OR 並列性 1. に比して独立性は低い．

以上のような並列性の特質をふまえ，静的負荷分散方式には OR 並列性 2. を，動的負荷分散方式には OR 並列性 1. を利用する．以下，次章と次々章で静的/動的負荷分散の具体的な実現方法について述べる．

4. 静的負荷分散方式

前章の考察に基づき今回採用した，PAX の OR 並列性を利用した負荷分散の具体的な方法について述べる．またその AP1000 版 KLIC への実装における評価結果を示す．

4.1 OR 並列性に基づく静的負荷分散

3.3節で述べたような理由から，OR 並列性 2 を利用した静的負荷分散を考える．具体的には各プロセスは自分が利用可能なプロセッサ範囲に関する情報を保持しており，子プロセスを起動するときにその範囲を子プロセスに等分して分配する．この方法ではプロセッサの株分けを行うときに一度通信を行えばその後は通信の必要がないため，通信回数が少なくて済む．また AND 並列に対応する処理は，1 プロセッサ上で逐次的に行うものとする．

PAX の説明に用いた図 3と同じ例を OR 並列に基づく静的負荷分散方式で処理した例が図 5である．$t = 1$ で PE$_i$ から PE$_{i+5}$ の 6 つのプロセッサを利用可能範囲として与えられているプロセス `v` は，$t = 2$ で子プロセスにそれらを分配する．例えば `np` を受け取ったときは子プロセスとして `v_np`, `vp1` の 2 つを起動するので，そのそれぞれに 3 つづつのプロセッサ PE$_i$ ～ PE$_{i+2}$, PE$_{i+3}$ ～ PE$_{i+5}$ を利用可能範囲として与える．`obj` を受け取ったときも同様に，3 つの子プロセス `v_obj`, `vp2`, `pred` に 2 つづつプロセッサを割り当てる．各プロセスは自分が利用可能なプロセッサ範囲のうち番号が最小のプロセッサ上で実行される．

表 1　性能評価に用いた文

	単語数	解の数	補強項なし
(1)	11	1	36
(2)	20	3	約 760

表 2　文 (1) の解探索に要した時間 (sec)

PE 台数	2	4	8	11	16
オリジナル	48.20	36.34	34.77	26.15	26.17
通信低減版	7.97	7.15	7.07	6.37	6.21
OR 並列静的	4.46	3.21	5.87	6.14	4.93

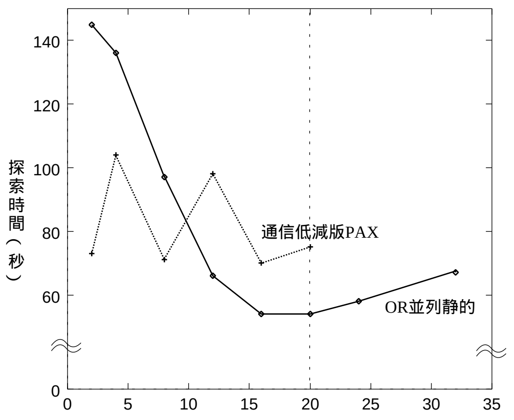
この例では受け取るデータが `np`, `obj` の 2 つなのでプロセスの起動は 2 回に分けて行われ，プロセッサの割り当てが 2 重にオーバラップして行われる．このようにこの方式の問題点は，受け取るデータが複数ある場合にその数に応じてプロセッサの割り当てが多重化してしまう点である．

4.2 性能評価

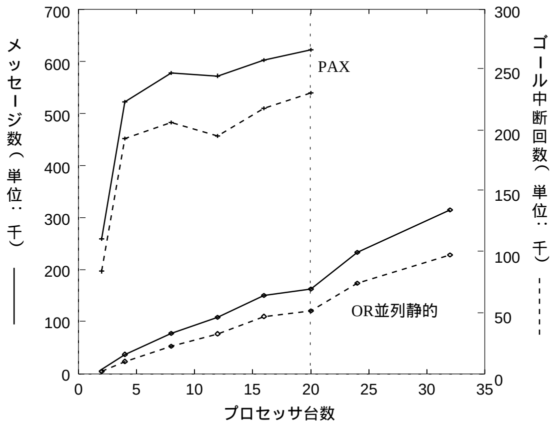
前節で述べた，OR 並列による静的負荷分散方式を導入した PAX プログラムを，当研究室で移植を行った AP1000 版 KLIC 上に実装し評価した結果を示す．KLIC[9),10)]は，ICOT が開発した KL1 処理系である．

今回の性能評価で用いたのは，文法数約 500，単語 (終端記号) 数約 200 からなる構文規則である．また，解析対象として用いた文は表 1に挙げた 2 つである．探索空間の広さを示すため，補強項を用いない場合に得られる解の個数も示す．

まず各 PAX プログラムに文 (1) を解析させ，その所要時間を測定した．解探索に要した時間を 表 2に示す．オリジナル PAX と通信低減版 PAX は単語数以上のプロセッサは利用しないため，プロセッサ台数が



(a) 探索に要した時間



(b) メッセージ数（実線）とゴール中断回数（破線）

図 6　文 (2) による評価結果

単語数を越えると実行時間はほぼ一定となっている．

更に詳しく効果を調べるため，通信低減版 PAX と OR 並列による静的負荷分散 PAX について文 (2) を用いて評価を行った．その結果を図 **6**に示す．

PAX は

- 再負荷分散を行わない．
- 利用可能なプロセッサ数が m であるとき，起動時に n 番目の単語を n mod m 番目のプロセッサに割り当てるため，$n > m$ である場合，負荷の偏りが生じる．

といった負荷の偏りを発生させる可能性を含んでいるため，実行時間は単調に減少するとは限らない．

これに対し OR 並列による静的負荷分散版 PAX はプロセサ数に応じてある程度単調に実行速度が向上している．また，各プロセッサ間のデータ依存性が軽減され，ゴール中断回数とメッセージ数が大きく減少している．メッセージ数がプロセッサ台数に対して線形に増加しているのは，第 4.1節で述べたように分散時に一度だけ通信を行うからである．

5. 動的負荷分散方式の提案

前章で述べた静的負荷分散方式は負荷分散のための処理が少なくて済むが，オリジナルと同様再負荷分散を行わない．構文解析における解の探索問題では探索木が対称性のある形になることは稀であるため，これでは負荷にばらつきが生じると考えられる．これを解消する方法として，master-slave 方式による動的負荷分散を提案する．なお，この動的負荷分散方式の実装は今後行う予定である．

5.1 概　　要

master-slave は，仕事の分配を行う master プロセスが 1 つ存在し，そのプロセスが各 slave プロセスに適宜仕事を割り当ててゆく方式である．

この方式を用いるときの問題点としては，master の仕事がネックになることが予想される．これを回避するため master の仕事をゴール転送先の紹介にとどめ，実際の仕事のやり取りは slave 同士の間で行わせる．こうすることによって master-slave 間の通信を抑えることができる．またこの方式では，slave 間の通信とは無関係に master はノード紹介を行えるため，slave 同士の間の通信はオーバラップされ，効率がよい．具体的な実現手法について以下に述べる．

各 slave はそれぞれ master との間に 2 本のストリームをもつ．一方は idle 状態になったときに自らのプロセッサ番号を master に送るためのものであり，もう一方はゴールの投げ先が必要となったときにその投げ先を示す未具体化変数を master に送るためのものである．負荷分散の単位となる投げるゴールは，3.3で述べた通り，OR 並列性 1. の関係にあるプロセスとする．これは KL1 プログラムのレベルで言えば，プロセス存続のための再帰呼びを送信することに対応する．

master はこれらのストリームをマージして得られた 2 つのキューの先頭要素から順次同一化していくことにより，未具体化変数であったゴールの送信先があるノード番号で具体化され，ゴールが実際に送信されることになる．

5.2 slave プロセスの実現

プロセッサが idle 状態になったときに自分のノード番号を master に送るための slave 上プロセス (以下，`tellidle` とする) の実現方法を考える．

先ず考えられるのは，担当するゴールの実行が終了した時点で `tellidle` を起動する方法である．この方法では確実に idle である時にのみ `tellidle` が起動される．但しゴールが中断してしまっている間 slave はただ参照変数が具体化されるのを待っているのみで，仕事は行わない．また OR 並列のみでなく AND/OR 両並列性を併用して負荷分散を行おうとした場合には，deadlock の危険性が生じる．これはゴール中断が発生

したとき，その原因を担う変数の具体化を行うプロセスが queue 内に残っている可能性があるためである．

次に考えられるのは，KL1 の `@priority` プラグマを用いて予め `tellidle` を最低優先度で起動しておく方法である．この場合は実行可能なゴールが他に存在しない場合はすぐに `tellidle` が実行されるので，先の方法のようなプロセッサ利用効率の低下や deadlock の危険性を回避することができる．但しデータ通信等が行われている間に `tellidle` が起動されてしまう可能性があり，余分に仕事を割り当てられてしまうことがある．

これら 2 つの方法では，後者の方が AND/OR 並列にも利用でき実行効率もよいものと思われる．

5.3 idle PE queue 操作の実現

あるノードにおいて実行可能なゴールがなくなると，あらかじめ最低優先度で起動されていた `tellidle` が起動され master にノード番号が送られるが，そののち `tellidle` は新しい仕事 (ゴール) がそのノードに割り当てられるまで wait 状態に入る必要がある．これは，idle 信号を何度も送ってしまうことで余分な仕事を割り当てられることを避けるためである．そしてその wait はノードに新たな仕事が割り当てられた後に解除されねばならない．

これを実現する方法として，以下のような手法を考える．`tellidle` は idle 信号を master に送ったのち一旦消滅するが，その後要求に対し新しいゴールが到着したときに，新たな `tellidle` が最低優先度で再起動される．

6. おわりに

本稿では，並列構文解析システム PAX の OR 並列性を利用した静的負荷分散法について述べ，その有効性を AP1000 版 KLIC 上で示した．OR 並列性を用いたことによってプロセッサ間のデータ依存関係を軽減することができ，結果として KL1 プログラムのゴール中断回数とメッセージ量が減ることで高い並列性を引き出すことができた．

構文解析の結果として得られる解の個数は少ないものであるが，各解は構文木を表現しているためある程度の大きさを持つ．PAX は解が解析の過程で 1 つのプロセッサに集まるようプロセスがマッピングされているが，OR 並列で並列処理を行った場合，解析終了時に解を特定プロセッサ上に集める必要がある．今後の課題としては，より効率を上げるために，この解の収集方法にも工夫が必要であろう．

動的負荷分散については，第 4章で述べた OR 並列性に基づく静的負荷分散に較べてプロセッサ稼働率が向上するが，その反面負荷分散のための処理と通信回数が増加する．負荷分散処理の速度は master と各 slave との間のストリームを通じた変数のやり取りに左右されると考えられるので，いずれにしても通信速度がネックになると考えられる．しかし粒度制御を行って細粒度に過ぎないように調節することで，通信オーバヘッドの過剰な増加を防ぐことができるであろう．

また，ゴール中断回数の減少を見ても明らかなように，OR 並列性を利用しているためプロセス起動時にその入力データがほぼ決定している．よって KLIC の一括送信[11)]機能を利用すると通信オーバヘッドを更に減少させせることができると考えられる．これは，処理効率がより通信速度に依存する master-slave にも適用してみる価値があるであろう．

参考文献

1) Matsumoto, Y.: A Parallel Parsing System for Natural Language Analysis, *New Generation Computing*, オーム社 (1987).

2) Chikayama, T.: *Introduction to KL1* (1994).

3) 佐藤裕幸: 並列自然言語構文解析システム PAX の改良, *Proceedings of KL1 Programming Workshop* (1990).

4) Kay, M.: Algorithm Schemata and Data Structures in Syntactic Processing, Technical Report CSL-80-12, Xerox PARC (1980).

5) Aho, A. and Ullman, J.: The Theory of Parsing, Translation, and Compiling, *Parsing*, Vol. 1, Prentice-Hall (1972).

6) Okumura, A. and Matsumoto, Y.: Parallel Programming with Layered Streams, *Proc. International Symposium on Logic Programming*, pp. 224–232 (1987).

7) 松本裕治, 奥村晃: 並列論理型言語による探索問題のプログラミング —層状ストリーム法の拡張—, 情報処理学会論文誌, Vol. 32, No. 7 (1991).

8) Pereira, F. and Warren, D.: Definite clause grammars for language analysis – A survey of the formalism and a comparison with augmented transition networks, *Artificial Intelligence*, Vol. 13 (1980).

9) 近山隆: KLIC ユーザーズ マニュアル (1995).

10) 六沢一昭, 仲瀬明彦, 近山隆, 藤瀬哲朗: KLIC 分散メモリ処理系の設計と初期評価, 並列処理シンポジウム JSPP '95 論文集 (1995).

11) 伊川雅彦, 大野和彦, 森眞一郎, 中島浩, 富田眞治: 並列論理型言語処理系 KLIC における通信の高速化 (1994).