Internal Use Only(非公開)

TR-I-0282

機能分割とパイプラインによる機械翻訳システムの並列処理手法

大橋　弘嗣
Hiroshi Ohashi

1992. 9

概要

自然言語処理における並列手法が色々提案されているが、そのほとんどがパーサーの並列化手法である。機械翻訳システムの並列処理の報告は少ない。本報告では変換主導型機械翻訳システムの並列化手法を並列手法のデータ並列、機能分割、パイプラインの内、機能分割とパイプライン処理の組み合せによる並列効果を述べる。また、この並列効果についてプロトタイプシステムによる実験結果を示す。プロトタイプシステムは密結合並列マシンーSequent S2000 <intel 80486 (25MHz) 16cpu>上で並列Lisp言語 allegro Clipで記述されている。なお、本報告で説明する変換主導型機械翻訳システムは現在文構造の解析処理が付け加えられており、翻訳処理の完成度はプロトタイプよりも高い。

ATR自動翻訳電話研究所
ATR Interpreting Telephony Research Laboratories

© ATR Interpreting Telephony Research Laboratories

１．目的

自然言語処理における並列化手法が色々提案されているが、そのほとんどがパーサーの並列化手法である。　機械翻訳システムの並列処理の報告は少ない。本報告では変換主導型機械翻訳システムの並列化手法を並列手法のデータ並列、機能分割、パイプラインの内、機能分割とパイプライン処理の組み合せによる並列効果を述べる。また、この並列効果についてプロトタイプシステムによる実験結果を示す。プロトタイプシステムは密結合並列マシンーＳｅｑｕｅｎｔ　Ｓ２０００　＜ｉｎｔｅｌ　８０４８６（２５ＭＨｚ）１６ｃｐｕ＞上で並列Ｌｉｓｐ言語　ａｌｌｅｇｒｏ　Ｃｌｉｐで記述されている。なお、本報告で説明する変換主導型機械翻訳システムは現在文構造の解析処理が付け加えられており、翻訳処理の完成度はプロトタイプよりも高い。

２．並列処理の概要

本報告での並列化手法は機能分割とパイプラインの併用により構築されている。機能分割とはある機能を持った処理別に処理構造を分割し、処理データを機能単位に分割して処理することである。パイプラインは機能処理を単位処理データの時間差により時分割に処理が可能になる。以下に逐次及び並列変換主導型機械翻訳システムの処理形態、及び並列処理形態の詳細を示す。

ａ．　逐次変換主導型機械翻訳システムの処理形態

変換主導型機械翻訳システムでは形態素解析で形態素切りされた日本語のストリングと形態素情報をもとに、変換知識情報から有効な変換知識を探索し複数の日本語構造候補を生成する。生成された日本語構造候補は変換知識に記述されている用例により角川類語辞典による用例距離計算を施され、英語変換されて用例距離を付加された英語文として出力される。

ア．　形態素解析
イ．　日本語構造生成
ウ．　英語変換、用例計算
エ．　英語文生成

逐次処理はアー＞イー＞ウー＞エの順に全解候補単位で処理される。

ｂ．　並列変換主導型機械翻訳システムの処理形態

並列処理では形態素解析は逐次で、そこで処理された情報をもとに日本語構造生成、英語変換及び用例計算、英語文生成ごとに機能分割しパイプラインで時分割処理を行う。実験では処理形態を機能分割の方法、並列実行の方法等、数種類行っている。詳細は次項で説明する。

ア．　形態素解析（逐次）
イ．　日本語構造生成（機能分割）
ウ．　英語変換、用例計算（機能分割）
エ．　英語文生成（機能分割）

並列処理ではアー＞イー＞ウー＞エの順に解候補単位で処理される。

c．　並列処理における日本語構造の機能分割の詳細

日本語構造生成時に経験則により優先順位の高い変換知識を選び出すことにより、数種類の日本語構造を生成する。この処理において優先順位の高い変換知識の候補を、機能分割でライトウェイトプロセス（以下ＬＷＰと記す）化された上記変換知識から生成される日本語構造を生成プロセスにメイルボックスを通して送り、複数のＬＷＰを生成し並列処理を行う。日本語構造生成時に助詞（に、の、を）、補助動詞（下さい、ます）から構造を生成しようとした場合、日本語構造探索が多数発生し処理時間が増加する。変換知識（は～ています）の場合処理時間は少なくて済む。

以下で示す処理形態は”［形態素解析］－［日本語構造生成］－［英語変換、用例計算］－［英語文生成］”を示す。

ア．　１－１－１－１形態

機能分割によりＬＷＰ化されるプロセスは機能ごとに１プロセスとし、メイルボックスで機能をパイプライン化する。この形態はメイルボックスでの処理待ちにより並列効率は良くない。

イ．　１－１－Ｎ－Ｎ形態

日本語構造生成プロセスでの生成処理は逐次で実行し、分割された日本語構造が生成されるごとにメイルボックスを通して下流の英語変換、英語文生成ＬＷＰへ分割された日本語構造を受け渡す。下流プロセスが分割され並列効率は良くなるが日本語構造生成処理が逐次のため、優先変換知識が多くなると、この処理に処理時間が片寄ることになり、全体処理時間の半分以上を占めることになった。

ウ．　１－Ｎ－Ｎ－Ｎ形態

日本語構造生成プロセスをＬＷＰ化し、並列に生成された複数の日本語構造候補は、各々これも機能分割でＬＷＰ化された英語変換処理にメイルボックスを通して送られる。この処理において日本語構造生成プロセスがＣＰＵ数でパイプラインとなり、各々の日本語構造生成の処理時間に差異が生じる。以下、英語変換、英語文生成ＬＷＰに分割された日本語構造がメイルボックスを通して送られ時間差で各々の日本語構造が処理される。

ウ－１．　Ｎ－Ｎ－Ｎ個のＬＷＰの動的生成

日本語構造、英語変換、英語文生成プロセスを日本語例文ごとの変換知識に応じて、日本語構造が生成されるごとに、以降のＬＷＰを動的生成し、日本語構造をメイルボックスに受け渡す。

ウ－２．　Ｎ－Ｎ－Ｎ個のＬＷＰの静的生成

処理実行前に必要個の機能別ＬＷＰを生成しておき、各々のＬＷＰをプロセス間の相互通信により協調動作（起動、停止）させることで、生成オーバーヘッドを短縮する。

エ．　１－Ｎ－Ｎ形態

１－Ｎ－Ｎ－Ｎ形態の下流プロセスをＮ－Ｎ　－＞　Ｎプロセスにまとめ下流プロセスのＬＷＰ生成、割り当てオーバーヘッドを低減する。

d．　部分的な並列処理

ア．　分割された日本語構造の用例計算を下位の日本語構造でＬＷＰ化し、フォークージョインでループ分割処理する。

イ．　ループ分割では実際の処理の分割と処理結果の回収の分割に分け、処理を並列化する。

ウ．　ループ分割の問題点としてループ内ｌｉｓｔ処理別ループではループ内のｌｉｓｔがｌｉｓｔごとに処理数が異なり処理粒度が一定にならない場合最大ｌｉｓｔの処理時間に処理速度が左右される。

e．　ｇｌｏｂａｌメモリのｌｏｃａｌメモリへの変換

並列処理を行う場合にＬＩＳＰにおいて関数と関数のデータの受渡しにｇｌｏｂａｌメモリを使用することもあるが、メイルボックスを使用してローカルメモリへコピーし、ＬＷＰへデータを受け渡すことで並列処理の処理結果をＬＷＰ各々が独立して処理可能になる。処理が終了後、各々のＬＷＰより処理結果をメイルボックスを通して回収し、独立処理で得られたデータを一つにまとめる処理が必要となる。

ｇｌｏｂａｌメモリ上で処理すべき場合はメモリにロックを掛け排他制御を必要とする。

f．　ａｌｌｅｇｒｏ　ＣＬｉｐの機能

主な機能を以下に示す。

ア．　ＱＬＩＳＰ、ＰＡＲＡＬＬＥＬ　ＬＩＳＰ、ＳＰＵＲＬＩＳＰ、ＭＵＬＴＩＬＩＳＰ各々の並列処理機能を組み合せて使用可能である。

イ．　ＬＷＰの生成、消去、停止、起動

ＬＷＰとはｌｉｇｈｔ　ｗｅｉｇｈｔ　ｐｒｏｃｅｓｓの略で機能別に並列処理を行う処理単位。

ウ．　メイルボックスの生成、メッセージの送付、回収、カウント

メイルボックスはＬＷＰごとに独立に処理しようとするデータをＬＷＰに受渡したり、ＬＷＰから処理したデータを回収するための通信路。

エ．　メモリロック機能（スリープロック、スピンロック）

ＬＷＰ同士が処理のやり取りをするためにメイルボックス以外に共有メモリを使用するときに、ＬＷＰ同士のアクセスによるデータの整合性を保護するために排他制御を行う機能。

3．並列処理の実験結果

ア．　ＬＷＰ別実行時間グラフ（形態　１－Ｎ－Ｎ）

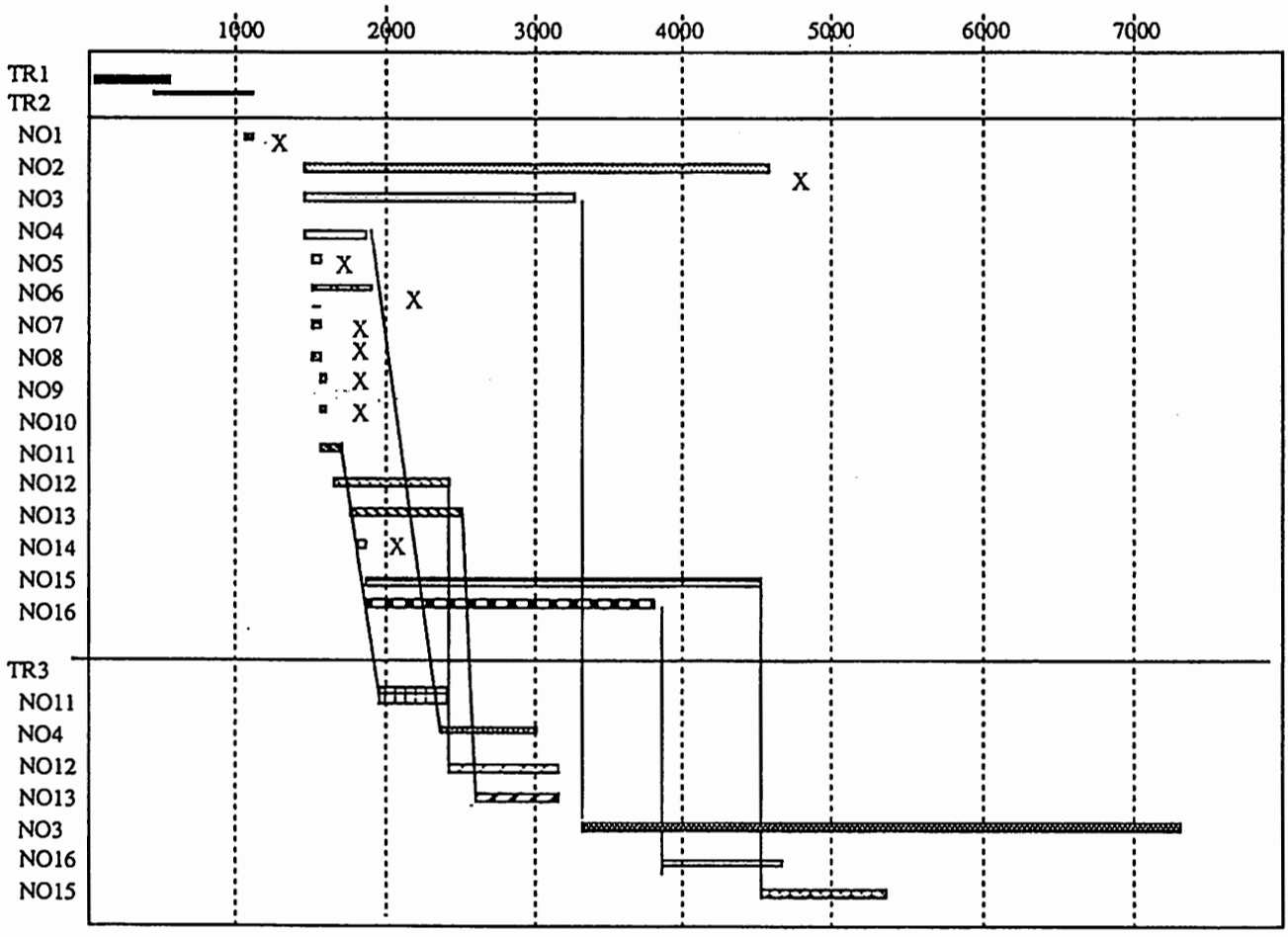
この例文では優先変換知識による日本語構造候補が１６候補作成され、そのうち７候補が構造的に可能であるため、英語変換、英語文生成（ＴＲ３）の処理が行われている。上記の優先変換知識１６候補は、以下に記す。

ＮＯ１	：ますと	××
ＮＯ２	：ます	×
ＮＯ３	：ています	○
ＮＯ４	：が～れています	×
ＮＯ５	：が	×
ＮＯ６	：と～ます	×
ＮＯ７	：は～ています	×
ＮＯ８	：は～れています	×
ＮＯ９	：は～れます	×
ＮＯ１０	：は～と～ます	×
ＮＯ１１	：には～れています	○
ＮＯ１２	：助詞ーが	○
ＮＯ１３	：助詞ーと	○
ＮＯ１４	：助詞ーは	×
ＮＯ１５	：助詞ーに	○
ＮＯ１６	：助詞ーには	○

○：日本語構造候補生成成功
×：　〃　失敗
××：初期段階失敗
（変換知識初期段階候補の条件選択削除）

この例文では処理時間がＮＯ３の”ています”の処理に制約されている。ＴＲ２及びＴＲ３の両処理とも他の処理の倍～５倍ほど掛かっている。ＴＲ２では構造探索の多さ、ＴＲ３では構造及び用例計算の多さに起因している。ＴＲ２の処理時間が増しても、生成構造が多いとは言えず、ＴＲ３の処理時間増に必ずしも結び付かない訳である。例えば、ＣＰＵを８個使っても”が～れています”と”には～れています”の処理は速く処理されるが、全体的には”ています”が終了しなければ全体の処理時間は速くならない。

（図１）



＊　例文：参加料には予稿集代と歓迎会費が含まれています（ＣＰＵ数：７）
＊　Ｘは日本語構造生成が失敗したことを示す。
＊　ｘ軸の単位：ｍｓｅｃ、ｙ軸はＬＷＰを示す。

＊　ＬＷＰ別用例距離値
日本語構造候補１６候補の内構造生成成功の７候補の用例距離値を以下に示す。

ＮＯ１１　：　１９
ＮＯ４　　：　１．６
ＮＯ１２　：　０．８３
ＮＯ１３　：　１９
ＮＯ３　　：　７．７７
ＮＯ１６　：　６．１６
ＮＯ１５　：　７．３４

イ．例文別ＣＰＵ数別処理時間

例文により並列度が異なるため、短い文で適用される優先変換知識が少ない場合、並列度は小さく、長い文で適用される変換知識が多い場合、並列度は大きくなる。また、助詞などは他種類の構造が探索されるため処理時間が長くなる。逆に、助詞＋変換知識の場合構造が特定され処理時間は短くなる。

（表１）

ＣＰＵ数／例文	例文１	例文２	例文３	例文４
１　real-time	２９．５３０	１６．１００	１．４７０	２．０３０
non-gc-time	２７．３３４	１５．５１７	１．０６６	２．０００
２	１７．４８０	１０．０７０	０．８３０	１．３９０
	１６．８００	９．２３０	０．８００	１．３１６
３	１２．９８０	８．７１０	０．７３０	１．１５０
	１１．８６７	７．７６６	０．７１７	１．１７０
４	１２．０００	８．４５０	０．７３０	１．１７０
	１０．４８３	７．２８３	０．７００	１．０８３
５	１２．１００	７．２００	０．７５０	１．１１０
	９．８９９	６．８３４	０．７５０	１．０８３
６	１２．９００	６．９８０	０．７４０	１．１００
	９．９５０	６．４３３	０．７３３	１．１００

＊　例文１：発表を希望されるのでしたら３月２０日までに要約を提出して下さい
例文２：参加料には予稿集代と歓迎会費が含まれています
例文３：そちらは会議事務局ですか
例文４：登録用紙で手続きをして下さい

＊　例文別優先変換知識候補

例文１：たら、れるのでしたら、下さい、助詞ーに、助詞ーまで、助詞ーまでに、れる、助詞ーを、
助詞ーの　（１０候補）

例文２：前記参照

例文３：です、ですか、は～です、は～ですか、助詞ーは　（５候補）

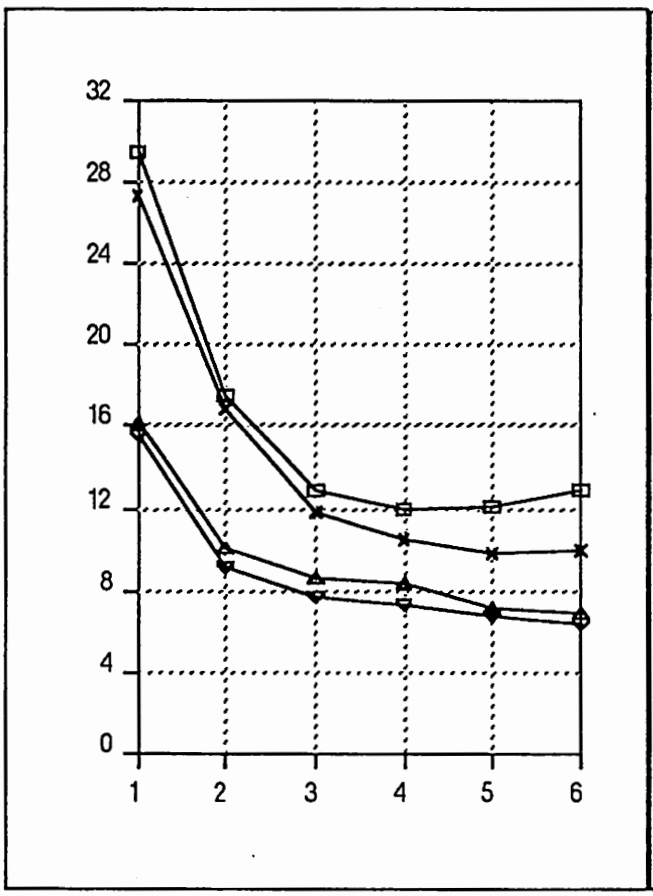
例文４：下さい、て下さい、で下さい、助詞ーを、助詞ーで　（５候補）

＊　例文別日本語構造生成時（ＴＲ２）の並列度、及び英語変換、英語文生成時（ＴＲ３(ＴＲ４、ＴＲ５)）の並列度

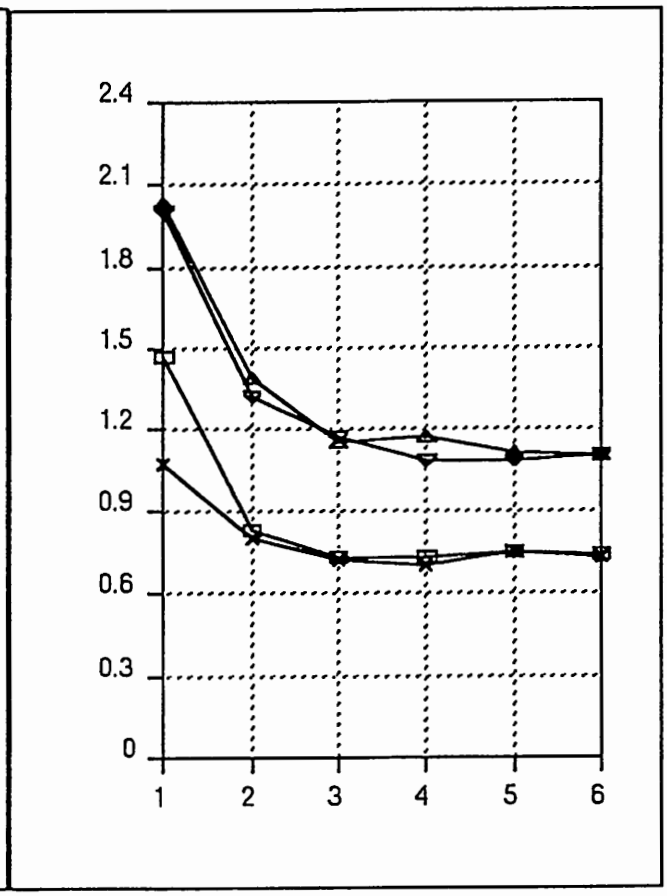
[１－Ｎ－Ｎ	例文１：	１０／６　文	[１－Ｎ－Ｎ－Ｎ	１０／６／６／６　文
形態]	例文２：	１６／７　文	形態]	１６／７／７／７　文
	例文３：	５／３　文		５／３／３／３　文
	例文４：	５／３　文		５／３／３／３　文

＊．処理単位　（ｓ）

（図２）



（図３）



＊　例文１：□　ｒｅａｌ－ｔｉｍｅ
×　ｎｏｎ－ｇｃ－ｔｉｍｅ
例文２：△　ｒｅａｌ－ｔｉｍｅ
▽　ｎｏｎ－ｇｃ－ｔｉｍｅ

例文３：□　ｒｅａｌ－ｔｉｍｅ
×　ｎｏｎ－ｇｃ－ｔｉｍｅ
例文４：△　ｒｅａｌ－ｔｉｍｅ
▽　ｎｏｎ－ｇｃ－ｔｉｍｅ

＊　処理単位　ｙ軸：（ｓ）　ｘ軸：ＣＰＵ数

４．まとめ

ア．　処理形態別に実験を行ったが密結合並列マシンでは処理粒度を細かくするよりも、処理粒度は粗い処理の方がメイルボックスでの通信処理時間に左右されにくい、処理粒度を細かくしマイクロ秒単位に同時通信を行うとバス競合が起こり通信処理時間に処理時間が左右されるが、１－Ｎ－Ｎ－Ｎ形態でも１－Ｎ－Ｎ形態でも粗粒度であるため通信処理時間に左右されていない。

イ．　助詞優先変換知識など処理時間の掛かる変換知識が含まれる文ではその処理に処理時間が制約を受け他の処理時間の短い処理の並列度増を阻害している。また処理時間の掛かる失敗構造生成が行われる文においても、その処理に処理時間が制約を受ける。

ウ．　先行制約制御依存

形態素解析－＞日本語構造生成－＞英語変換－＞英語文生成の処理プロセスにおいて、各々の処理プロセスが先行制約を受け、先行プロセスの終了を待って後続プロセスの処理が行われる。後続プロセスは先行プロセスの加工したデータを再度加工し、最終プロセスで結果を生成する。

解候補を分割することで日本語構造生成プロセス、英語変換プロセス、英語生成プロセスが同時並列に解候補別に並列処理されている。処理単位が解候補単位であり、粗粒度で他情報の制約を受けない構造であるため、処理待ちによる処理順序の制約は受けない。

後続プロセスが先行制約制御依存を受けなければ処理順序を入れ換える処理構造も考えられる。

エ．　先行制約データ依存

逐次処理では各々のプロセスで処理された解候補データをグローバルメモリに蓄え後続プロセスで処理していた。このため全解候補が生成されることが必要となり、解候補データの先行制約データ依存により後続処理は全解候補生成の処理時間の制約を受ける。

並列処理では解候補データを解候補単位に分割することにより、先行制約データ依存の処理時間制約を短縮するようにＬＷＰに振り分けている。

細粒度並列を行おうとすると処理待ちによる処理順序制約による先行制約データ依存が発生し、データの処理単位別整合性を保つ必要が出て処理構造は複雑になり、スケジューリング、バリア同期等が必要になる。

オ．　並列実行による成功可能性探索の早期発見と失敗可能性及び未処理探索の処理中止の問題

用例距離計算結果の決定値により、並行処理中または待ち行列中の未処理及び処理途中ライトウェイトプロセスを処理中止することで同時進行過程の処理時間の掛かる失敗探索や成功探索を処理しない方法を考察する。

用例距離計算の確からしさの許容範囲を０～３とすれば、”０”は、ＥＸＡＣＴ　ＭＡＣＴＨでそれ以上は近似値による構造生成であり、０～３までの結果が出力されれば、他の解候補は必要ではないともいえる。よって、並列処理中に０～３までの英語文が生成されれば、他の処理中の解候補はもはや必要とはいえない。この方法で処理時間の掛かる探索及び未処理の探索を処理中止すれば、大幅な処理時間の短縮となる。この方法は並列実行以外でも可能であるが、そのトリガーを発見するには並列処理が有効である。なぜなら、トリガーが最も最後に実行されることも起こりえるからであり、並列処理すれば同時に実行され早期発見に結び付く。

上記例文では許容範囲”０～３”では２番目に終了したＮＯ４を正解とするとｒｅａｌ－ｔｉｍｅで約３．０２秒で他処理を中止でき、許容範囲”０～１”では３番目に終了したＮＯ１２を正解とすると約３．２１秒で他処理を中止することが可能である。処理時間はＮＯ４では５．０倍、ＮＯ１２では５．

0倍にすることができる。ＣＰＵ８個ではＮＯ４で約２．６６秒で終了し６．０倍、ＮＯ１２は変らなかった。この処理はデータのみでありインプリメントはしていない。

参考文献

[１] 松本「並列構文解析」　自然言語処理　５３－２　（１９８６）
[２] 寿崎、佐藤他「マルチＰＳＩにおける並列構文解析プログラムＰＡＸの実現及び評価」　並列処理シンポジウム　ＪＳＰＰ’８９
[３] 加藤「素性構造の単一化に基づくパーサの並列化手法の効率」　ＡＴＲ　Ｔｅｃｈｎｉｃａｌ　Ｒｅｐｏｒｔ
[４] Ｈｉｄｅｔｏ　Ｔｏｍａｂｅｃｈｉ「Ｑｕａｓｉ－Ｄｅｓｔｒｕｃｔｉｖｅ　Ｇｒａｐｈ　Ｕｎｉｆｉｃａｔｉｏｎ」　ＡＴＲ　Ｔｅｃｈｎｉｃａｌ　Ｒｅｐｏｒｔ
[５] 藤岡、苫米地、古瀬、飯田「並列時間差準破壊型単一化アルゴリズムの実現手法」　ＡＴＲ　Ｔｅｃｈｎｉｃａｌ　Ｒｅｐｏｒｔ
[６] 古瀬、飯田「変換主導型機械翻訳の実現Ａ」　ＡＴＲ　Ｔｅｃｈｎｉｃａｌ　Ｒｅｐｏｒｔ
[７] 佐々木、神余、笠原、成田「原子力プラント状態予測シミュレータへの並列処理の適用」　並列処理シンポジウム　ＪＳＰＰ’８９
[８] 笠原、藤井、本多、成田「スタテック．マルチプロセッサ．スケジューリング．アルゴリズムを用いた常微分方程式求解の並列処理」　情報処理学会論文誌　ｖｏｌ．２８　Ｎｏ．１０　（１９８７）
[９] 坂井「並列計算機におけるスケジューリングと負荷分散」　情報処理　ｖｏｌ．２７　Ｎｏ．９　（１９８６）
[１０] 喜連川、大城「共有メモり型並列マシン（ＳｙｍｍｅｔｒｙＳ８１）とＳＩＭＤ型超並列マシン（ＭａｓＰａｒ　ＭＰ－１）の連続鋳造設備モールド部温度計算への適用並びにその評価」
[１１] 宮野「並列化とその限界ー理論的側面から」　コンピューターソフトウェア　ｖｏｌ．７　Ｎｏ．１（１９９０）
[１２] 「Ｓｙｍｍｅｔｒｙ　Ｔｅｃｈｎｉｃａｌ　Ｓｕｍｍａｒｙ」　Ｓｅｑｕｅｎｔ社
[１３] 「Ａｌｌｅｇｒｏ　ＣＬｉＰ　Ｕｓｅｒ　Ｇｕｉｄｅ」　Ｆｒａｎｚ社
[１４] 笠原「並列処理技術」　コロナ社
[１５] 高橋「並列処理機構」　丸善

xx. 並列変換主導型機械翻訳システムの起動方法

0。 シークエントS－2000 に ログイン する。

1．NEMACSを起動する。

2．M－x clipを入力する。

clip file の load
パラレル file の load
漢字テーブル の load

<initial lwp> が 表示される。

3．形態素解析プログラムのロード

:ld /usr2/ohashi/ppatran/morph/setup

4．並列patran の ロード

:ld /usr2/ohashi/ppatran/setup

5．パッケージの指定

(in－package 'ppatran)

6．TRANSLATEの方法

(translate "会議に申し込みたいのですが")

.
.
.

＊ 国際会議予約の例文は /usr2/ohashi/ppatran/a－text を参照。

7．結果の表示

(pprint *output*)

8．lwp別プロセスreal－timeの表示

(lwp－run－time)

9．機能処理別プロセス数の表示

（ｌｗｐ－ｃｏｕｎｔ）

１０．ＣＰＵ数のｓｅｔ

（ｓｅｔ－ｎｕｍｐｒｏｃｓ　ＣＰＵ数）　：ＣＰＵ数　１～１５

１１．エラーの対処

ｌｗｐの途中でエラーになったとき

．

．

エラー表示

＜ｌｗｐ３２＞　．．．エラーの原因のｌｗｐ

：ｒｅｓ　ｒｅｓｅｔ

：ｌｏｃ　ローカルエリアパラメター表示

：ｚｏ　　ゾーンエリアパラメター表示

反応が返ってこない時

ｃ－ｃ　ｃ－ｃ　ｌｉｓｐ　ｉｎｔｅｒｕｐｔ

ｃ－ｘ　ｃ－ｃ　強制終了

１２．ＣＬＩＰプロセスが　ｐｓ　ｕｘ　コマンドで表示されｋｉｌｌされていないとき

ＫＩＬＬ－ＣＬＩＰ

１３．ｍｏｎｉｔｏｒの表示

ｃｄ　／ｕｓｒ／ｅｔｃ　ディレクトリーで　ｍｏｎｉｔｏｒ　を起動する。

１４．ｃｏｍｐｉｌｅの方法

：ｃｌ　ファイル名