1B−7

並列構文解析アルゴリズムの密結合マシン上への実装と評価

鄭 基泰*　中藤 哲也*　峯 恒憲**　谷口倫一郎*　雨宮真人*
* 九州大学大学院総合理工学研究科
** 九州大学教養部

1 はじめに

我々は、実時間自然言語理解システムの構築を目指して、まず解析時間 $O(n)$・プロセッサ数 $O(n^2)$ の性能を持つ並列構文解析アルゴリズムを提案し[1]、次に構文解析と形態素解析の協調動作を行うシステムの提案を行ってきた[2]。今回は、この協調型の並列形態素・構文解析システムを共有メモリ型の並列計算機上に実装し、特に、共有メモリに対する排他制御法についての実験を行ったので、その報告を行う。

2 並列形態素解析と並列構文解析の協調動作の概要

構文解析は、LR構文解析法をベースとし、文脈自由文法から構成されたLR状態遷移図を解析の制御に使用する。文法規則は文節間の係受け関係を記述したものであり、終端記号を文節としている。一方、形態素解析では正規規則から構成された文節ネットワークを使用し、これをたどって文節を構成する。構文解析と形態素解析とのやりとりはshift動作の時に行い、そのインタフェースには、文節を格納するためのテーブル(以下、文節テーブルと呼ぶ)を利用している。

まず構文解析で生成されたプロセス(構文プロセスと呼ぶ)が、ある入力文の位置(SP)からのshift動作を行う場合、文節テーブルに、構文プロセスの情報(sxinfo)を書き込む。もし、そのテーブルに既に文節が格納されていれば、その文節に対してshift動作を続ける。まだ、文節が格納されていない場合、形態素解析部に入力文の位置(SP)を渡し、SPから抽出できる文節の情報を要求する。形態素解析部では、文節を構成する途中、抽出された単語の品詞や読みなどの種類、文法情報の種類の数だけ、形態素プロセスを生成する。各形態素プロセスは文節ネットワークをたどりながら、文節情報を構成できたとき、文節テーブルにその文節情報を書き込み、そこに書いてあるsxinfoのshift動作を続行させる。

3 共有メモリ型計算機上での負荷分散

共有メモリ型の並列計算機では、複数のプロセスによる共有変数への競合が起こる。その共有変数に対する排他制御のためのLOCKがボトルネックになりやすい。そのため、LOCKの部分の負荷を抑える必要がある。また、プロセスの粒度が小さい場合はプロセス生成によるオーバヘッドが生じるためプロセス生成を抑制する必要もある。

3.1 排他制御の分散方法

インプリメントする方針として、重複処理を避けるために大域変数(共有変数)を利用し、なるべくプロセス間での不用なコピーを避けるようにした。例えば、形態素解析などで抽出した文節情報は共有変数である文節テーブルに保存し、複数のプロセス間で共用される。しかし、共有変数が一つで排他制御のためのLOCKも一つ(図1-a)の場合、あるいは共有変数の配列要素に一つのLOCK(図1-b)しか持ってない場合に共有変数の競合はに起こりやすい。これらの場合はLOCKによる負荷は避けられない。そのため共有変数とLOCKを次のように分散する。

1. 形態素解析によって抽出された文節情報を格納する文節テーブル(BUNSETSU-TABLE)などの共有変数は配列とする。
2. 排他制御のためのLOCK変数(BUNSETSU-TABLE-LOCK)を配列として実現する。
3. BUNSETSU-TABLE[SPk]にアクセスしてデータを書き込む時は、インデックス[SPk]を持つBUNSETSU-TABLE-LOCK[SPk]を取る。LOCKが取れないプロセスはLOCKが取れるまで待つ。この方法によって共有変数の分散排他制御を行なう。

3.2 排他制御の分散LOCK

共有変数とLOCKを配列で分散するため配列要素分LOCKの負荷が分散される。例えば、BUNSETSU-TABLEにアクセスする時、図1の(a)と(b)の場合はLOCKの部分がボトルネックになる。図1の(c)の場合は各BUNSETSU-TABLEの配列要素ごとにLOCKがあるので別々の共有変数にアクセスするような形になる。その手順を示すと図2のようになる。図2のA,Bで共有変数とLOCK用の配列を用意し、Cではを N 個生成して、用意した配列にいれる。Dでk要素のLOCKを取って、EでCRITICAL SECTIONに入り、Fでk要素のLOCKを返す。

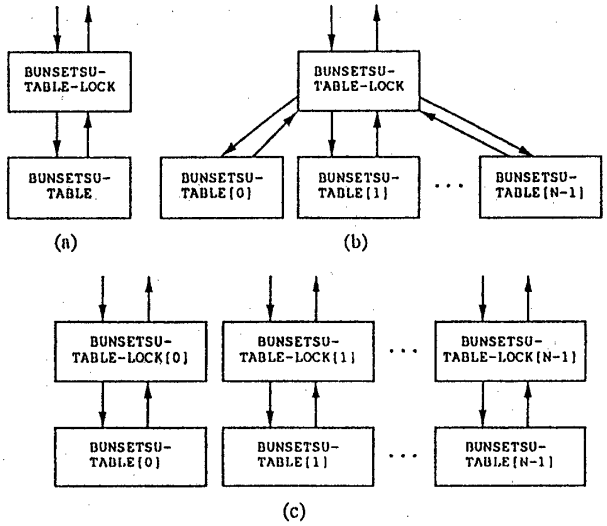

図1 LOCKと分散LOCK

An Implementation of Parallel Morphological and Syntactic Analysis Algorithm in Shared-Memory Multiprocessors and Its Evaluation

Kitae Jung, Tetsuya Nakatoh, Tsunenori Mine, Rin-ichiro Taniguchi and Makoto Amamiya
Department of Information System, Graduate School of Engineering Sciences, Kyushu University

A:(BUNSETSU-TABLE 用の配列を作る)
B:(BUNSETSU-TABLE-LOCK 用の配列を作る)
C:(BUNSETSU-TABLE-LOCK の各インデックスごとに
別々の LOCK を作って入れる)

D:(BUNSETSU-TABLE-LOCK のインデックス k に入っている
LOCK を獲得する)
E:(BUNSETSU-TABLE-ARRAY のインデックス k の値を取る)
E:(BUNSETSU-TABLE-ARRAY のインデックス k の値を変え
る)
F:(BUNSETSU-TABLE-LOCK のインデックス k に入っている
LOCK を返す)

図 2 分散 LOCK の制御

3.3 分散排他制御する時の LOCK の制約と問題点

- 共有変数でインデックスを使って参照出来ない場合は不可能である。即ち、LOCK と共有変数にアクセスするためのインデックスが必要である
- 共有変数のインデックス単位では競合が起こる。
- 共有変数のあるインデックスに競合が集中する時の負荷の分散は制御できない。
- LOCK にアクセスされる回数によって性能が異なる。粒度が粗く、共有変数へのアクセスが負荷がかからない 場合は効果がない。
- LOCK と UNLOCK の間で実行される CRITICAL SECTION の負荷によって性能が異なる。
- プログラムの粒度の大きさによって性能が異なる。
- PE(Processing Element) の数によって影響を受ける。

3.4 プロセスの生成の抑制

我々のアルゴリズムでは、実行中複数の曖昧さが出る場合はその数分のプロセスを生成する。そのプロセスの生成の仕方はほとんどが図 3 のようなトランスファータイプで行なわれる。しかし、一つのプロセスの部分の粒度が小さい時はプロセス生成のオーバヘッドが大きくなるためその場合は子プロセスの最初のプロセスは生成せずに親プロセスが引続き次の処理をやる方がよい。これにより細粒度プロセスの過多な生成を抑制し、プロセス生成のオーバヘッドを抑えることができる。

図 3 でプロセス P0 は二つの子プロセスを生成せず、最初の子プロセスは親プロセス P0 が引続き処理する。二番目の子プロセスのみ生成する。プロセス P1 は次に子プロセスを生成する時はプロセス P3 のみ生成する。図 4 は図 3 を記述した例である。KEITAISO のプロセスが P0 の場合、A 部分のプロセスは図 3 のプロセス P1 に割当てる。B 部分のプロセスは図 3 のプロセス P0 が引続き実行する。次の C 部分のプロセスは P2、P3 プロセスに、D 部分のプロセスが P0,P1 のプロセスに割当てる。

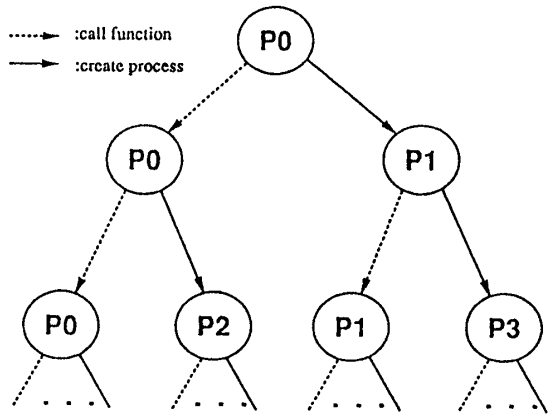

図 3 プロセス生成の流れ

(defun KEITAISO (入力 SP)
(入力の文の SP から得られる単語を wordlist に入れる)
(wordlist の最初の単語を word1 に入れる)
(wordlist の中で word1 以外の単語を wordrest に入れる)
A:(wordrest の単語分 (HINSHI) を呼ぶプロセスを生成する)
B:(親プロセスは word1 を関数 (HINSHI) に渡す。)

(defun HINSHI (word)
(一つの単語は多数の品詞を持っている場合もある。
その中最初の品詞を hinshi1 に入れる。)
(多数の品詞の中 hinshi1 の以外は hinshirest に入れる)
C:(hinshirest 分 (文節ネット) を呼ぶプロセスを生成する)
D:(親プロセスは hinshi1 を関数 (文節ネット) に渡す)

図 4 プロセスの抑制生成

4 まとめ

実験に使用している並列計算機は i80486 の CPU を 20 個搭載した共有メモリ型並列計算機 Sequent S2000 で使用言語は Allegro CLIP である。例文は科学技術論文や種々の本から無作為に選んだ約 40 個の文を使用し、辞書は約 2000 語程度のものを使用して行なっている。現在は今回述べた排他制御の負荷分散処理法、細粒度プロセス生成の抑制法などの評価実験を続けている。現在プロセスの台数効果は大体 3 − 6 倍くらい早くなった。今後は今のアルゴリズムを分散メモリ型並列計算機に適するように修正し、64 台の CPU を持つ分散メモリ型並列計算機 AP1000 上に実装することを計画中である。

参考文献

[1] 峯, 谷口, 雨宮: 一般の文脈自由文法に対する効率的な並列構文解析情報処理学会論文誌, 第 32 巻, 10 号

[2] Mine, Taniguchi, Amamiya: Coordinated Morphological and Syntactic Analysis of Japanese Language. Proc. of IJCAI-91, pp. 1012-1017, 1991.

[3] 渡辺: 並列処理解説, コロナ社, 1991