アーリー法による並列構文解析プログラムの実現

2F−8

下山朋彦，早川栄一，並木美太郎，高橋延匡
（東京農工大学　工学部　電子情報工学科）

１．はじめに

並列プログラムを作成する際には、タスクに仕事をどう割り振るか、タスク間で共有するデータをどう扱うかが問題になる。我々はこの問題を考察するために、自然言語の構文解析法であるアーリー法を並列プログラムとして実現し、評価を行った。

２．アーリー法のアルゴリズム

アーリー法は曖昧な文法を許す自然言語向けの構文解析手法である。

アーリー法ではｎ個の集合を中心に処理を進める。集合の要素は構文解析の途中結果を示す。初期状態では集合０以外は空集合であり、処理が進むにつれて集合１，２，...にも要素が含まれるようになる。

途中結果を集合として持つため、構文木の作成は集合への操作として行われる。集合０～ｎのそれぞれに、次の三つの操作を行う。この様子を図１示す。

操作１：集合ｋの中に条件に合う要素があれば、その要素を加工して集合ｋにつけ加える。

操作２：集合ｋの中に条件に合う要素があれば、集合ｊ(ｊ＜ｋ)にも条件に合う要素があるか探す。もし集合ｊにも条件に合う要素があれば、その要素を加工して集合ｋにつけ加える。

操作３：集合ｋの中に条件に合う要素があれば、その要素を加工して集合ｋ＋１につけ加える。

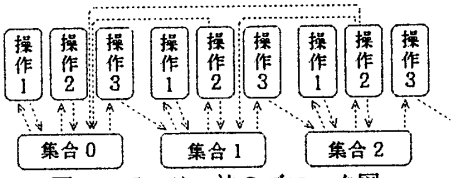

図１．アーリー法のブロック図

３．並列化の方法

２章で示したアルゴリズムを三種類の方法で並列プログラムとして実現した。それぞれの並列化の方法を示す。

３．１　並列化の戦略

一つのタスクに割り当てる仕事の単位は、操作１～３とした。これは、仕事の単位として考えやすいことと、もし十分な並列性が得られなくても、各操作内部を並列化して並列性を増すことができると考えたためである。

各操作を並列化する方法として、一つの操作に一つのタスクを割り当てる方法と、一つの操作に複数のタスクを割り当てる方法を考えた。

３．２　一操作一タスクによる並列化〈共有メモリ版〉

図１の操作１～３に一つずつタスクを割り当てることで、並列化を行った。このときすべての集合は、すべてのタスクで参照可能な共有メモリに置く。これは操作２，３は、自分の属している集合以外のタスクへも参照を行うためである。この様子を図２に示す。

すべてのタスクが共有メモリを頻繁にアクセスするため、共有メモリのアクセス競合が予想される。

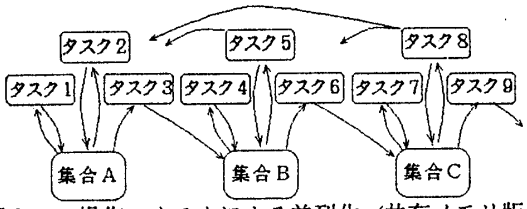

図２．一操作一タスクによる並列化〈共有メモリ版〉

３．３　一操作一タスクによる並列化〈メッセージ版〉

３．２では、集合をすべてのタスクで共有するために、メモリ競合が問題になった。そこで、集合を各集合に属する三つのタスクだけで共有することで、メモリ競合を緩和することを考えた。この様子を図３に示す。

自分が属していない集合にアクセスしたい場合は、メッセージにより、その集合に属するタスクにアクセスを依頼する。属していない集合への参照コストは大きくなるが、各操作は自分の属する集合以外はあまり参照しないと予想される。

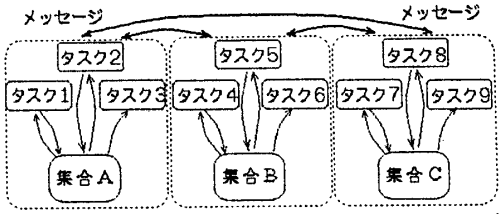

図３．一操作一タスクによる並列化〈メッセージ版〉

３．３　一操作に複数タスクを割り当てる並列化

集合内の一つ一つの要素にタスクを割り当てることで並列化を行う。新しい要素をつけ加えるときには、一緒にその要素を処理するためのタスクを生成する。タスクはその要素の処理が終わると終了する。この様子を図４に示す。

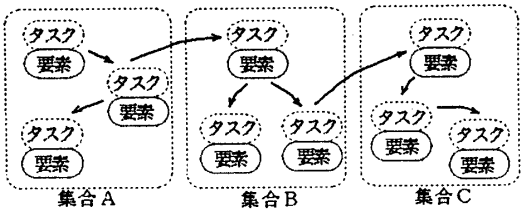

図４．一つの操作を複数のタスクで行う並列化

An Implementation of Parallel Parsing Programs with the Earley Method
Tomohiko SHIMOYAMA, Eiichi HAYAKAWA, Mitarou NAMIKI, Nobumasa TAKAHASHI
Tokyo University of Agriculture and Technology.

4．評価

三つの実現方法を、タスクへの仕事の分割方法、集合の共有の方法という観点から評価した。

4．1　タスクへの仕事の分割方法

今回の実現では、一つの操作に一つのタスクを割り当てる並列化、一つの操作に複数のタスクを割り当てる並列化の、二つの方法で並列化を行った。

最初にアーリー法に含まれる並列性を調べ、理想的な並列化を行ったときの状態を示す。次に各並列化方法が、理想的な並列化の、どの部分を実現しているかを示す。

(1) 理想的な並列化

一つ一つの操作を並列化の基本単位と考え、各操作の理想的な並列化を考察した。

各操作は、条件にあった要素を探し、その要素から新しい要素を作成する。そこで、どの要素からどの要素が作成された、という依存関係のグラフを作ることができる。今回の評価のために使用したテストデータ（生成規則数 10，入力記号数 3）によって作られた要素の依存関係のグラフを図５に示す。グラフの矢印についている①～③は、矢印の示す処理が処理１～３のどれだったかを示している。

このグラフの互いに並行した矢印は、原理的に並列に処理できる操作を示す。これは、それらが互いの処理結果を利用することがないためである。また、それらの操作をすべて並列化したときが理想的な並列化である。

並行した矢印の数を調べることで、図５のテストデータで理想的な並列化を行ったとき、平均並列度 2、最大並列度 4 であることがわかった。

(2) 一つの操作に一つのタスクを割り当てる並列化

図５を見ると、操作２によって得られた要素を操作１が処理し、操作１によって得られた要素を操作３が処理する、といったサイクルがある。そのため、一つの操作に一つのタスクを割り当てると、操作１による要素を操作２が処理している間に、別の要素を操作１が処理するような、パイプライン処理が期待できる。

このパイプライン化により、実際にどのくらい並列化されたかを、図５で用いたテストデータにより調べた。平均並列度 1.45、最大並列度 2 となり、理想的な並列化を行ったときに比べてかなり低い並列度となった。また、このとき生成されたタスクの数は 12 である。

(3) 一つの操作に複数のタスクを割り付けた並列化

一つの操作に複数のタスクを割り付けた並列化では、新しい要素が作られると、要素に対応したタスクが作成される。そのため、要素の生成とタスクの生成は同じになり、タスクのダイヤグラムは図５と同じになる。アーリー法の並列性を最大限に生かしており、平均並列度 2、最大並列度 4 であることがわかる。ただし、このとき生成されていたタスクの数は、要素の数と同じ 32 である。

(4) 二種類の並列化の評価

二種類の並列化のどちらも、並列度の割に生成されたタスクの数が多い。一つの操作に一つのタスクを割り当てた並列化は、本質的にはパイプライン処理である。そのため、もっと規模の大きな入力データを使用した場合には、もう少し並列度が上がるかもしれない。しかし、一つの要素に一つのタスクを割り当てる並列化では、無駄なタスクを生成することは避けられない。図５を見ると、一つの要素から複数の要素が作成されることは稀である。そのため、要素ごとにタスクを割り当てても処理が並列化されない。評価結果を検討し、一つのタスクに割り当てる仕事を見直す必要がある。

4．2　集合の共有の仕方

３．３で述べたように、その集合に属するタスクだけで集合を共有する実現は、各タスクは主に自分の属している集合を参照するという予想から行った。

この予想を確かめるために、タスクが属していない集合を参照する回数を計測した。結果を表１に示す。この表から属している集合といない集合へのアクセス回数の比がおよそ 4:1 であることがわかった。どちらの方法が有利かは、使用するシステムにおいてメモリ競合の深刻さと属していない集合のアクセスコストにより決まる範囲であり、どちらの方法も妥当な実現であるといえる。

表１．属していない集合を参照する回数

	操作１	操作２	操作３
属している集合	76	239	54
属していない集合	0	82	4

5．おわりに

コーディングの段階では妥当に思えた並列化の戦略も、詳しい評価を行ってみるとあまり並列性が得られないこともあり、評価の重要性を認識した。

パイプライン処理による並列化の効果を評価するためには、動的な処理の流れや、データの依存性を調べる必要があった。しかし動的な処理の流れのトレースは非常に手間がかかる。動的な処理の流れを解析し、並列化の効果を調べる評価系を作成することが必要である。

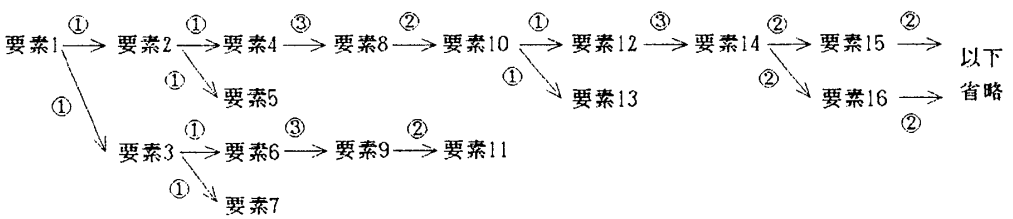

図５．要素の依存関係