TAO-Lindaの実現

2E−8

明石　修
NTT ソフトウェア研究所

1　はじめに

Lindaは、通信する相手を意識する必要がない、ポータブルな並行プログラミング機能を提供することを目的として作られた言語である。[1)]

しかし、現在はその特徴的な操作に注目し、疎結合、密結合マシンを問わずに並行計算を行なう計算モデルとして捉えられている。実現は、既存の言語に、Lindaモデルに特有な操作を加え、並行プログラミング機能を持った新しい派生言語を作ることにより行なう。

本論文では、言語TAO[3)]の上に実現したTAO-Lindaに関して、その設計思想及び実現方法について述べ、評価を行なう。

2　Linda並行計算モデル

Lindaモデルには、全てのプロセスに共有される仮想的な一つの空間(tuple space、略してTS)が存在する。ユーザプロセスは、TSを介してオブジェクト(tuple)をやりとりし、生成的会話(generative communication)を行なうことにより、並行計算を実現する。[2)]

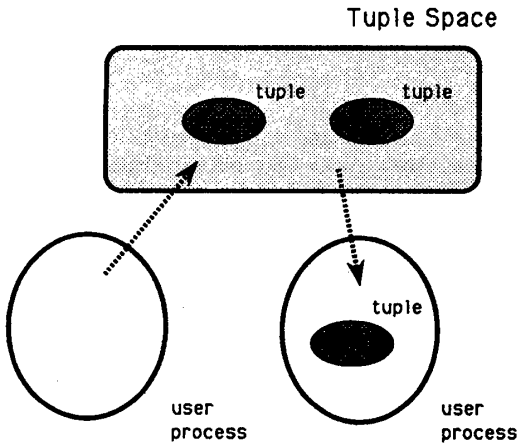

図1　Lindaの生成的会話

2.1　TSへのアクセスとprimitive

Lindaモデルでは、TSへのアクセスを、out、in、rd、evalの4つの操作で実現する。

1. out: tupleをTSに出す。
2. in/rd: tupleをTSから、取り出す/読む。引数として与える要素の値とマッチングを行い、マッチする候補の中から1つのtupleを非決定的に選ぶ。適当なtupleがない場合は、マッチするtupleがoutされるまでブロックされる。
3. eval: 別プロセスで評価を行い、その結果をtupleとして、TSに置く。

また、引数として変数を与えることができる。変数は、以下のように引数リストに"? 変数"と書く。

```
(out ... ? x ...)
```

変数はどの値ともマッチするが、変数と変数はマッチしない。また、rd、in操作時に変数を書いた場合、その変数にはマッチした値が代入される。

3　TAO-Lindaの仕様

TAO-Lindaでは、in、rd、out、eva* の4つの操作を提供する。現在のところ引数として使える型は、整数、文字、文字列、及びそれらのリストである。また、それらの型を返す関数を書くこともできる。

```
(out "nue" 1 '(1 2 ("elis" 5 6)))
(in "tao" (+ 1 2 3) 7)
```

in、rd、out操作の引数は、通常のlisp関数の引数と同様に、その操作の実行前にそのプロセス中で評価される。evaは、新しいプロセスを生成し、その中で引数を評価する。その後、outと同様にその結果をTSに置く。引数として関数が与えられた場合、eva操作は、別プロセス中でその関数を実行するが、これはLindaモデルにおいて、並列度を増していく手段となる。なお、"?"の次の変数は評価されない。

以下に、TAO-Lindaにおけるマッチングの例を示す。

out操作	マッチするin操作
(out "tao" 1 (+ 2 3) '(7 8))	(in "tao" 1 ? x ? y)) この後、xは5、yは(7 8)
(out "nue" ? x '(7 8))	(in "nue" 1 '(7 ? y)) この後では、yは8

表3-1 マッチングの例(TAO-Linda)

4　Lindaによるプログラミングと問題点

Lindaモデルは、計算を実行する環境を抽象化し、意識する必要がないように設計してある。しかし実行効率

Implementaion of TAO-Linda
Osamu Akashi (akashi@nuesun.ntt.jp)
NTT Software Laboratories

*lispのevalと区別するため。

を考えた場合、最適な粒度を決定するためには、モデルの範囲を越える部分まで考慮しなければならない。

考慮しなければならない点の第一は、通信コストである。粒度の細かい計算は、共有メモリをもつマルチプロセッサマシン上で実現したものであれば、より実行効率が上がり、粒度の大きいものであれば、分散環境上でもそれほど実行効率は落ちない。

第二は、TS へのアクセスのオーバーヘッドである。抽象度を高めたことにより、プログラマが考慮しなければならない部分は減少したが、一つ一つの操作に TS でのマッチング、同期という部分が存在する。従って、粒度が細かくなれば、その分だけ TS をアクセスする操作が増え、オーバヘッドも増加する。

5 TAO-Linda の実現

実現は、言語 TAO を用いて、ネットワークで接続された複数の ELIS ワークステーション上で行なった。†

システムは、TS の情報を管理するサーバ (TS サーバ) と、ユーザプロセスからの TS へのアクセス要求や、TS サーバからのデータを管理するクライアント (TS クライアント) からなる。

サーバ、クライアント間の通信は、TCP/IP プロトコル群の UDP(User Datagram Protocol) をベースに行なった。具体的には、UDP パケットの中に管理情報とデータをエンコードし、パケットの再送によって信頼性を保証する独自のプロトコルを規定した。

単一サーバの場合、データ、負荷が集中し、それがボトルネックになりかねない。データの分散化を進める場合、処理を分散させることにより粒度の細かいものへの対応ができるが、実現は複雑となる。

この検討結果から、まず、単一サーバで実現し、その後は、複数サーバ方式に移行していくことにした。

6 評価

前述したように、実行効率を考えた場合、現在の TAO-Linda のような分散環境上での実現では、粒度の粗い計算方式が適していると思われる。その一つの例として、ある行列計算を逐次に実行する場合と、別々のマシン上に存在するクライアントに行ごとに計算を依頼し、その演算結果を後で集める場合に分けて、時間測定を行なった。その結果を図に示す。

Linda モデルのオーバヘッドは、通信のコスト、tuple の生成、マッチングの機構である。その結果、6×6 行列の計算の時は 10%、12×12 行列の時は 18% 速度が低下した。

†将来的には、マルチプロセッサ構成の MacElis-II でも単一プロセッサでも動かす予定である。

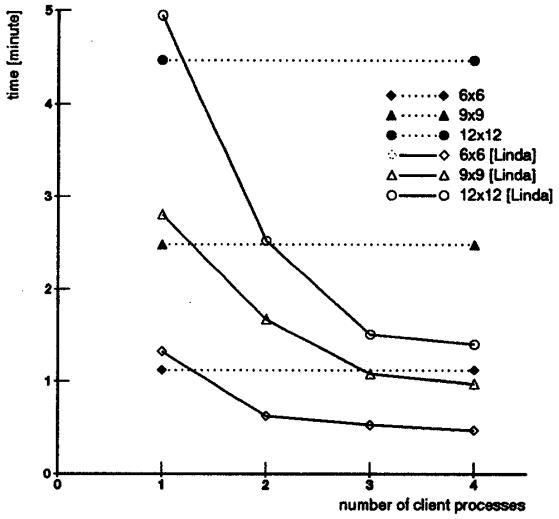

図 2 行列演算の実行結果

クライアントの数が 2 個の時は、6×6 行列が逐次計算の 1.7 倍、12×12 行列は、1.8 倍速度が向上した。クライアントの数が 4 個の時は、6×6 行列が逐次計算の 2.4 倍、12×12 行列は 3.2 倍速度が向上した。

しかし、クライアントが増えていくと、通信負荷やサーバへの負荷が増加するため、速度の変化は鈍くなり、ある一定数を越えるとそれ以上実行速度は速くならない。

7 まとめ

分散環境上に UDP をベースに Linda モデルを実現した。その上で、時間測定評価を行ない、粒度の大きな計算の場合には、有効であることがわかった。

この時、独自のプロトコルを規定したが、この同期の部分はかなり冗長であり、高速化のための改善が必要である。また、今後データの分散化を進めていくが、この分散アルゴリズムが全体の性能に大きく影響してくるので、この部分も今後の課題である。

最後に、貴重なご意見を頂いた NTT ソフトウェア研究所の村上主任研究員、天海研究主任、そして研究の機会を与えて頂いた奥乃主幹研究員に感謝します。

参 考 文 献

1) David Gelernter 他, Distributed Communication via Global Buffer, In Proceedings ACM Symp. on PODC, Aug. 1983

2) Gelernter, Generative Communication in Linda, ACM TOPLAS, Jan. 1985

3) Ikuo Takeuchi 他, A List Processing Language TAO with Multiple Programming Paradigm, New Generation Computing Vol.4 Num.4 1986