2M−6

Lindaモデルにおける Tuple Spaceの構造化*

野里貴仁 杉本明 阿部茂
三菱電機株式会社 中央研究所

1 はじめに

Lindaは既存のシステムに通信プリミティブを付加することにより並列システムを生成する。その特徴は Tuple Spaceを使った通信にある。Tuple Spaceを使うことで、複製作業者モデルを容易に実現できる。しかし、全プロセスに共有される Tuple Spaceの存在は大規模プログラムの作成時にモジュラリティの点で問題がある。本論文では、Tuple Spaceとプロセスを一まとめにすることで Tuple Spaceの構造化を提案する。

2 Linda

Linda[1, 2] は Tuple Space(TS) を使った交信を行なうことで並列計算を行なうモデルである。ここで Tupleとは名前と値の組である。Lindaではこの Tupleを Tuple Spaceと呼ばれる仮想的な共有メモリにアクセスすることでプロセス間の通信を行なう。Tupleに対する操作としては in、out、read、evalの4つの操作がある。out は TS にデータを置き、in は TS からデータを取り除く、read は TS からデータを読むだけで、取り除くことはしない。in、read では Tuple にパターンマッチしたものが取り除かれる / 読まれる。この時 Tuple の中の仮引数は対応する値を代入される。例えば、(''foo'', 10, 100) が TS にある時に in(''foo'',?x, ?y) を実行すると x、y にはそれぞれ 10、100 が代入される。(?x は x が仮引数であることを示す。)eval は別のプロセスで実行した結果が Tuple として、TS に残る。

2.1 複製作業者モデル (replicated worker model)

これは均質的なデータに対し同一実行主体を複数個作り、各プロセッサに割り当て、処理をするモデルである。実行主体 (作業者) はデータの集合から自律的にデータを取り込み、処理を行なう。データがなくなれば作業は終了する。例えば、行列の掛け算では行と列を掛け合わせるプロセスが行と列を取り出し、計算を行なうことを繰り返す。

このモデルの次のような利点を持つ。

負荷分散 各作業者は動的に仕事を取り出すので、1つのプロセッサに大きな負荷がかかってしまうことはおきにくい。また、動的にプロセスを生成する場合にも対応できる。

scalability プロセッサの数が増えればそれに応じてスピードが上がることが期待できる。

フォールトトレランス 各作業者はそれぞれ独立に作業をするので、一台のプロセッサの故障がシステム全体の故障にはつながらない。

複製作業者モデルは Lindaを使うことで容易に実現できる。

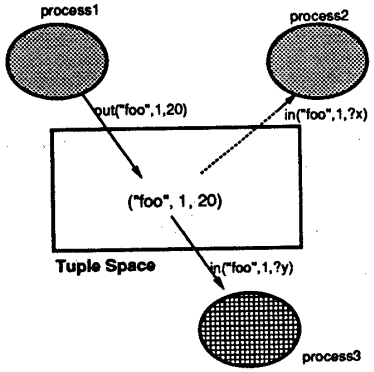

図 1: TS の問題点

2.2 Lindaの問題点

Lindaの問題点は単一の TS をすべてのプロセスが共有していることである。このために大規模なプログラムを作ることは困難になる。プロセスは何の制限もなく TS アクセスすることできるので、予期せぬ副作用をもたらす可能性がある。例えば、process2 が in("foo",1, ?x) を実行してブロックされている時に、別のプロセス process3 が、間違えて in("foo", 1, ?y) を実行して、tuple ("foo",1,20) を取り除いてしまうと計算が正しく進まない。最悪の場合はプロセスはブロックされたままである (図 1)。

また、大規模システムでは既存のサーバプロセスなどを使わなければいけない。これらを直接 TS と交信させることは容易ではない。

3 TS オブジェクト

最初に挙げた問題は Tuple に対して、情報隠蔽が行なわれていないことが原因であった。process1、process2 が扱っている Tuple を process3 から見えないようにできれば、問題は起こらない。

そのようにするためには TS を分割し、関連するプロセスのみがアクセスできるようにする。TS とプロセスを1つにまとめることで、このようなことが実現できる (図 2)。また、システム全体では TS とプロセスがまとまったもの同士が、互いに通信をすることが必要である。また、二番目の問題は TS を使う以外に通信手段がないことが原因であった。これには他の通信手段を組み込むことで対処できる。

以上のことから、TS オブジェクトを提案する。これは TS とプロセスを一つにしたオブジェクトを考え、システムがこのオブジェクトから構成される世界である。TS オブジェクトが互いに通信を行なうことで計算が進む。

Structuring of Tuple Space in Linda Model
Takhito NOZATO,Akira SUGIMOTO,Shigeru ABE
Mitsubishi Electric Corp.

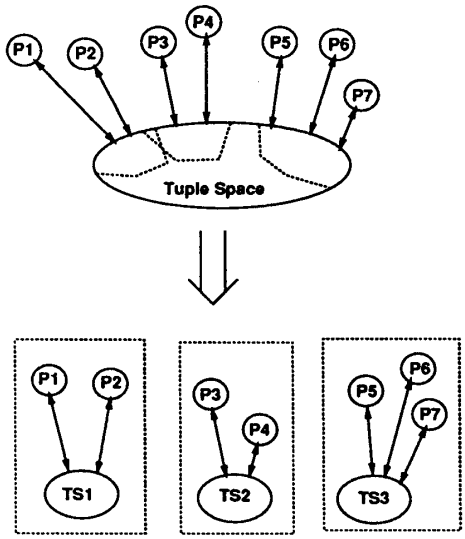

図 2: TS の構造化

3.1 TS オブジェクト

TS オブジェクトは TS とそれにアクセスするプロセスから構成される。この TS に直接アクセスできるプロセスはオブジェクト内のプロセスだけで、オブジェクト外のプロセスからは直接アクセスすることはできない。これによって TS への不法なアクセスを禁止する。

TS オブジェクトの中のプロセスは常に動いているか、TS へデータを待っているかのどちらかの状態にある。動いているプロセスが複数個あっても良い。TS へ Tuple を置くことによりプロセスは自らデータを取り出し、処理を始める。

3.2 オブジェクト内通信

オブジェクト内の並列実行の単位であるプロセス同士の通信は TS を使った通信となる。これは Linda で使われる TS を使った通信と全く同じである。これによって複製作業者モデルをオブジェクト内で実現することができる。

3.3 オブジェクト間通信

システム全体から見れば、オブジェクト間の通信はメッセージのやりとりによって行なわれるように見える。オブジェクトの内部では外部から来たメッセージは Tuple に変換されて、TS に置かれる。

呼び出し側ではオブジェクトへのメッセージ送信には send、get の 2 つのプリミティブを用いる。

send メッセージを Tuple に変換し、送り先の TS に置く。この時、送り先のオブジェクトで一意な ID を返す。この ID はオブジェクトに対するリクエストを表す。send の実行後すぐに次の実行に移る。

get 引数にオブジェクトと ID を指定し、対応するリクエストの結果をオブジェクトに要求する。リクエストの結果がまだ求められていない場合は結果が求まるまで呼び出し側の実行は止められる。結果が既に求まっていれば、この結果をメッセージに変換して返す。

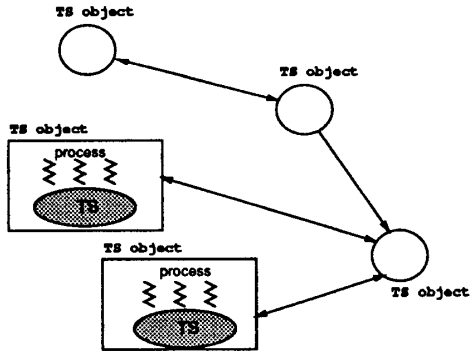

図 3: オブジェクト間の通信

受け手側では send によって、out(“request”, id, ...) が実行され、Tuple が TS に置かれたように見える。プロセスはこの Tuple を取り出し (読み込んで)、計算を開始する。get によって、in(“reply”,id,?result) が実行され、対応する Tuple を取り除かれるように見える。

呼び出し側と受け手側の対応は次のようになる。

呼び出し側	受け手側
id=send(object,...)	in(“request_id”,?id) out(“request_id”,inc(id)) out(“request”, id, ...)
get(id,object,result)	in(“reply”, id, ?result)

オブジェクト間の通信のモードについては、ABCL/1[3] では次に 3 つに分類している。

- 過去型。メッセージ送信後、オブジェクトは次の実行に移る。
- 現在型。メッセージ送信後、返答が返ってくるまで、オブジェクトは待つ。
- 未来型。future を使った過去型メッセージ送信。

この 3 つのモードは send と get のプリミティブを組み合わせて使うことにより容易に実現できる。

4 まとめ

複製作業者モデルを中心に Linda モデルの改良を試みた。グローバルな Tuple Space を分割し、TS にアクセスするプロセスと一まとまりにした TS プロセスを考え、不法な TS へのアクセスを制限した。また、TS とプロセスをまとめたことにより、クライアントサーバのサーバなどを複製作業者モデルによって実現することができる。

参考文献

[1] Sudhir Ahuja 他,Linda and Freinds, IEEE Computer, Aug. 1986

[2] Nicholas Carriero 他,Linda in context, Communication of ACM, Apr. 1989

[3] 米澤明憲 他, オブジェクト指向に基づく並列情報処理モデル ABCM/1 とその記述言語 ABCL/1, コンピュータソフトウェア,Vol.3, No.3., 1986

[4] Antonio Corradi 他, Parallelism in object-oriented programming languages,International Conference on Computer Language,1990