並列オブジェクト指向言語DinnerBell

データフロー実装向きの最適化

6U-8

河野真治・明石考祐・田中英彦
東京大学工学部

DinnerBellは，オブジェクト指向と単一代入則に基づくデータフローの概念を結びつけることを目的として設計された言語である。これまで，オブジェクト指向言語の実行形式であるメッセージパッシングを，さらに引数毎に分解した実行単位を考え，それをプロセッサに割り当てる方法を提案してきた。

この方法による実装は単純ではあるが，冗長な部分があることが，シミュレータの実装の過程で明らかになってきた。ここでは，その冗長な部分をのぞき，より効率的に並列実行を行う方法について考察する。

1.DinnerBellの実行

DinnerBellの実行単位は，メッセージ送信を引数毎に分解したもので，ここではイベントまたは，コンテクストと呼ぶ。イベントは，メッセージを構成する①セレクタと（ただ一つの）②引数の組，③送信先，④答を返すべき変数（Continuation），⑤次の引数とのリンク（replySelf と，ここでは呼ぶ）の五つの部分からなる。このイベントを各プロセッサの分配することにより，DinnerBellでは，並列実行を実現する。

各プロセッサでは，このイベントをプールし，一つ一つ処理していく。この時に，オブジェクトのインスタンスの生成と，一時変数の確保をイベントの管理とは，別に行う。イベントの処理により，以下のようなことが行われる。

①イベントの生成
②一時変数の生成
③インスタンスの生成
④答の送信

この時に，オブジェクトのインスタンスの生成と，一時変数の確保をイベントの管理とは，別に行う。現在の実装では，インスタンスの持つデータ領域（あるいは環境）は，プロセッサ固有で移動しない。したがって，プロセッサ間での通信は，イベントか，または，答の送信である。これにより，DinnerBellの実装では，次の二つのフローが存在する。

①メッセージ送信――――コントロール・フロー
②答の転送――――――――データフロー

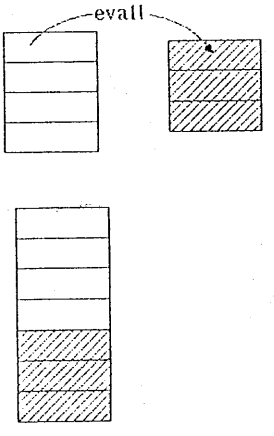

図1　余計なメッセージ送信の除去

2.最適化

DinnerBellの最適化は，これら二つのフローにそって行う。現在は，以下の4つの方法を考えている。

① 余計なメッセージの除去。引数なしのメッセ

```
$class Factor |
  fact:N □ ~R. mul:1 to:N:
  mul:M to:N □
    (N=:1)
      yes:(□ (~R□) ret:M)
      no: (□ mul:(M*:N)to:(N-:1))
|
```

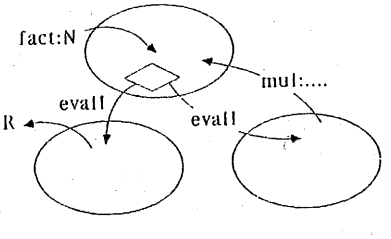

図2　階乗の計算の例

Concurrent Object Oriendted Language DinnerBell - optimiztions for Data-flow execution

Shinji Kono, Takahiro Akashi, Hidehiko Tanaka
The university of Tokyo, Faculty of Engineering

ージの場合，メッセージ送信先のクラスが前もって分っていれば，そのメッセージを省略して，そのメッセージを受け取る先のメッセージ送信を起動してよい（図1）。DinnerBellでは，クラス定義の中に，他のオブジェクトからは見えないクラスを多く作り，その間の通信は，引数なしのメッセージを使うことが多いので，この最適化が有効となる。

② コントロール・フローの再利用。同じオブジェクトに繰返しメッセージを送信する場合，一時変数（methodVarible）は，メッセージ送信ごとに作成されるが，インスタンス変数（itBlockVariable）は，同じ値が使われる。この時もし，ループがあれば，同じメッセージ送信が繰り返されることになる。既に発火することの分っているイベントと，新しく生成したmethodVaribleを組にすることにより，一度転送したイベントを再利用することができる。図2は，メッセージパッシングで構成した階乗のループを使った計算である。ここでmul：というメッセージがループを構成するが，実際にメッセージ送信する代りに，MとNの値をもつ一時変数を引き渡すだけで計算を進めることができる。

③ if文の最適化。図2の例題では，if文が使われている。DinnerBellでの現在の実装は，TRUEとFALSEの二つのクラスに対するブロック（オブジェクト）を転送し，TRUEまたは，FASLE が，そのブロックに対して，eval！を送信する方法をとっている。この方法では，ブロックの送信と，eval！の送信が，if文一つについて行われる。一つの方法は，条件式の評価が終るまで待ち，その後，yesパートとnoパートのイベントを起動すれば，よい。しかしDinnerBellでは，単一代入則を採用しているので，yesパートとnoパートを同時に実行し，正しいパートの環境のみを，条件判断の結果に従って，外部に接続するということができる。これには，制限があって，このif文の中の変数の値が，副作用を持つオブジェクトであってはならない。DinnerBellでは，変数自身は，単一代入であるが，その値となるオブジェクトは，システム関係など副作用を持つことがある。この方法をとるかどうかは，ユーザが選択する。これは，基本的には，if文の先読みであり，これと，前のイベントの再利用を行うことより，DinnerBellは，データ駆動なだけでなく，データフロー実行を行うということができる。

④ オブジェクトのデータ構造に基づく実行。オブジェクトは，例えば集合などの様に，それ自身で，並列性をもっている場合がある。このような場合は，そのオブジェクトのデータ構造にそって並列実行することが可能である。例として，集合の要素に対して，すべて同じ動作をするとする（図3）。この時，基本的には，集合の各要素に対してメッセージを送信するのだが，そのメッセージを省略して，集合演算そのものを行うことができる。これは，もちろんハードウェアのサポートがなければ実現できない。

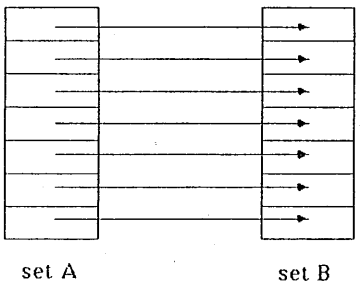

図3 集合を表すオブジェクトの並列実行

3.関数型言語との違い

他の並列オブジェクト指向言語と異なりDinnerBellは，もともと副作用のあるオブジェクトがなければ，言語仕様上は，ただひとつメッセージのＪｏｉｎを除いて，副作用は存在しない。言語としてピュアである。この範囲では，言語の表現能力としては，副作用のない関数型言語と同じである。しかし，実装の方法や，最適化の方針は，大幅に異なる。これは，オブジェクト指向言語が次のような特徴をもっているからである。

タイプ・フリー
モジュール化
メッセージ・パッシング

オブジェクト指向言語の特徴の一つは，変数にタイプがないことである。変数に代入される値は，実行時になるまで予測できない。したがって，オブジェクト指向言語では，特にユーザに指定した場合と自分自身へのメッセージ送信の場合以外は，タイプに関する最適化は難しい。

オブジェクト指向言語の基本は，プログラムのモジュール分割にある。これは一方では，プログラム全体に関する情報をコンパイル時に得られないことを意味する。例えば，あるメッセージ・セレクタがシステム全体でユニークかどうかを判定することはできない。このような点からオブジェクト指向言語の最適化は，実行時の情報を用いる必要がある。

最後にメッセージパッシング自身は，送信先のオブジェクトに対する間接呼出しとみなすことがでのきる。しかし，間接呼出しの対象となるオブジェクトの種類が非常に多いのが，問題となる。この部分に関しては，メソッド・ハッシングなどの方法が知られている。

DinnerBellの実装では，メッセージを引数単位に分解して送るために特に，メッセージパッシングそのものを減らす最適化が重要である。