2E-7

分散処理システム記述言語の設計とそのカーネルの製作

大和　理、田中　英彦、元岡　達
(東京大学工学部)

1. はじめに

最近、分散処理記述用言語の研究がいくつか進められている。[1][2][3]

本稿では、主記憶を共有しない複数のプロセッサから構成される分散処理システム記述に適した言語(DIPROL)の設計について報告する。

2. 分散処理記述言語の課題

主記憶を共有しない分散処理システムは、集中型システムと比べて、次のような特徴がある。

(1) メッセージ転送を用いるためプロセッサ間の通信コストが高い
(2) プロセッサによって、データの内部表現が異なる場合がある
(3) 集中的なシステム管理が難しい

(2)については、標準のデータ表現に変換しての転送を考える必要がある。

言語設計における問題点は、

(A) プロセス間の相互作用の表現
(B) 相手プロセスの指定の方法
(C) メッセージのタイプ・チェック
(D) 分割コンパイル
(E) カーネルの実行効率
(F) エラー・ハンドリング

(A)には、プロセス・コールによる方法[2]とランデブによる方法[1]がある。前者はメッセージ転送によって指定されたプロセスが、新たに生成されて処理を行なう。後者は、プロセスが相互に通信要求を出し合い、それが一致すれば、メッセージ転送を行ない、それぞれが必要な処理をする。(B)については、プロセス名という文字列だけでは、相手がユニークに決まらない可能性がある。(E)では、標準データ表現への変換や、メッセージ転送手順を行なうカーネルの処理速度が実用上問題になる。

3. 言語の設計

設計の方針としては、

(1) 容易にインプリメントできること
(2) Hoare[4]のいう、非決定性を含むメッセージ転送を実現すること
(3) プロセス間通信などの機能分散に適していること
(4) 各プロセッサごとの分散管理が容易なこと

基本言語としてPASCALを選び、(1)の理由から、中間言語インタプリティブ方式とし、さまざまな拡張を行なった。(3)の機能分散の一つの例として、当研究室で設計製作した、網向きプロセス間通信制御プロセッサ(CUPID)[5][6]を用いたシステム構成を考え、プロセス間の相互作用の表現には、ランデブによる方法を用いた。

主な拡張について、次に説明する。

(A) バッファ・タイプ宣言

転送するメッセージのデータ構造をPASCALのレコード型のように宣言する。互いに通信するプロセスは同一のバッファ・タイプ宣言をもっていることが必要である。

(B) プロセス・ラベル宣言

ライブラリ関数のような場合、結果を送る相手名は決まらない。そこで、プロセス・ラベルを用いて指定することにした。

(C) エントリ宣言

メッセージ通信の入口・出口の名前を宣言する。プログラム・ボディでは、このエントリ名を呼びだすことにより通信が行なわれる。

```
entry  NAME=send( x,y )
       NAME( z,u,w )
  x : communication parameter
  y : buffer type identifier
  z : process designator
  u : result variable
  w : message list
```

Wというメッセージをzで指定されるプロセスに対して送る。Wがyなるバッファ・タイプ名で示されるデータ構造であるかどうかをコンパイラがチェックする。ランデブの成立は、バッファ・タイプの比較により決定される。

(D) 標準プロシージャの追加

通信終了を待つために *waitcom*、プロセス・ラベルに実プロセス名を割り当てるために *regist*、プロセスの生成などを行なうために、*start*、*create*、*stop*、*remove* がある。

(E) *alternate* 文

あらかじめ、どんな通信が行われるかわからない場合の処理を記述するために *alternate* 文を加えた。これは、複数のプロセスからランダムに送られる種々の要求に応じた処理の記述に適している。いずれかのエントリでランデブが起こると、あとに続く *statement* 列の処理をするもので、*until* のうしろの *expression* が真になるまでくり返される。ランデブの成立は、CUPID において決まるので終了条件が真になっても、すでに受信したメッセージの処理をすべて行なった後に、次へ進む。

4. カーネルの製作

カーネルは、現在製作中であるが、基本的には、C-PASCAL のカーネルにメッセージ・ハンドラの部分をつけ加えることになる。CUPID に渡すコマンドとパラメータは15Bほどで、これらの領域は、実行開始前にあらかじめ用意し、パラメータをできるだけコンパイル時にセットすることで実行効率の向上を企っている。メッセージのキューイングはCUPID が行なう。通信エラーは、CUPID が検出してホストに知らせ、メッセージ・ハンドラはその原因を、*result variable* にセットする。プロセス番号は、プロセッサ番号とローカル・プロセス番号とを合成したもので決まる。各ローカル・プロセス番号は、プロセスが生成される時にローカルに決定されるので、通信する相手のプロセス番号は、実行開始直前にプロセス・ヘッダ上のテーブルにセットされる。

```
buffer  TAB1=structure  integer: 5  end; -----バッファ・タイプ宣言
        TAB2=structure  integer: 5  end;
        TAB3=structure  integer: 2  end;
process USERRECTAB;
  entry  STREG=receiveany(BLOCK,TAB1);----エントリ宣言
         ENDREG=receiveany(BLOCK,TAB2);
         LIST=receiveany(BLOCK,TAB3);
  prlabel #RESO; ----- プロセス・ラベル宣言
  type  TIME=record HOUR,MINUTE,SECOND: integer;  end;
        その他のタイプ宣言
  var  START,ENDT: TIME; RESONO,USERNO: integer;
        その他の変数宣言
  begin
        テーブルのイニシャライズ
    alternate
      STREG(#RESO,RESULT,USERNO,START,RESONO):
          ユーザが使うリソース番号とその使用開始時間を記録
      ENDREG(#RESO,RESULT,USERNO,ENDT,RESONO):
          ユーザの使用時間を求めて、終了時間とともに記録
      LIST(#RESO,RESULT,USERNO,RESONO):
          ユーザが、あるリソースをいつ、どれだけ使ったかを
                                          リストする
    until  FALSE; (ずっとくり返す)
  end;
```

プログラム例

どんなユーザがどのリソースを、いつ、どれだけ使ったかを記録するプロセスの記述例

5. まとめ

機能分散を意識した言語設計によって実行効率のよいシステムプログラムの記述が可能になる。CUPID を用いないシステムにおいても、この設計はカーネルのモジュール化を進める上で有利である。今後は、実装を進めるとともに、応用プログラムを記述しながら、言語の改良を行なっていく予定である。

〔参考文献〕

(1) DoD : Preliminary Ada Reference Manual, ACM SINPLAN Notices, Vol. 14, No. 6, Jun. 1979
(2) Robert P. Cook : *MOD - A Language for Distributed Programming, IEEE, Vol. SE-6, No. 6, pp.563-571, Nov. 1980
(3) J.A. Feldman : High level programming for distributed computing, CACM, Vol. 22, No. 6, pp.353-368, Jun. 1979
(4) Hoare, C.A.R. : Communicating Sequential Processes, CACM, Vol. 21, No. 8, pp.666-677, Aug. 1978
(5) 和田、小森、田中、元岡：網向きプロセス間通信制御プロセッサのハードウェア，本大会論文集
(6) 和賀井、和田、田中、元岡：網向きプロセス間通信制御プロセッサのソフトウェア，本大会論文集