プログラミング言語 24－1
（1990. 2. 9）

並列プログラミング言語SERVE

廣谷良彰　福田晃　村上和彰　富田眞治
（九州大学大学院総合理工学研究科）

あらまし　並列プログラミング言語SERVEは，大規模マルチプロセッサシステムで実行されることを意図した並列処理問題記述用言語であり，科学技術計算を中心とする高並列で膨大な計算量を必要とする応用分野へ適用していくことを目的とする．SERVEでは中程度の粒度でメッセージ通信によって処理を行う．種々の並列処理形態が素直に記述できるように，メッセージ通信として同期／非同期通信，対多通信および同期通信を用いた並列呼出しなどが可能であり，また，並列処理の単位であるプロセスは静的／動的に生成し得る．本稿では，SERVEの言語機能，処理系および簡単な応用例について考察を混じえながら述べる．

The Parallel Programming Language SERVE

Yoshiaki HIROTANI, Akira FUKUDA, Kazuaki MURAKAMI, Shinji TOMITA
Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
6－1 Kasuga－Koen, Kasuga－shi, Fukuoka, 816 JAPAN
hirotani@kyu－is. is. kyushu－u. ac. jp

Abstract　SERVE is a parallel programming language on large－scale multiprocessors for applications with a large amount of parallelism and computation such as scientific computation. SERVE is suited for medium grain parallelism. Communication and synchronization in SERVE is accomplished only by a message－passing facility. In order to describe a broad class of parallel applications, various message communication facilities are provided. These include syncronous／asyncronous communications, ANY or ALL type communication and parallel call with syncronous communication. Processes are created as static or dynamic instance.

In this paper we describe SERVE language facilities, and its characteristics are discussed.

1. はじめに

膨大な計算量を必要とする応用分野は数多く存在し，計算機に対する高速化の要請は留まるところを知らない．この内高い並列性を内在する分野に対する解決策として，VLSI技術を背景にしたマルチプロセッサ構成による大規模な並列計算機が注目を集めている．そこで我々は，様々な並列処理形態に柔軟に適応できる”可変構造型並列計算機”とその並列オペレーテイング・システムの開発を進めている[1]－[6]．

高並列処理環境において重要な点は，解くべき問題の持つ並列性をソフトウェアレベルでどのように抽出するかである．この対処法は次の２つに分類される．

① 並列化コンパイラによる自動検出

② 並列言語による明示的な指定

①は，プログラム資産の継承および既存の逐次型言語の継続的使用が可能であるという点で有用である．しかし，逐次プログラムに基づくアルゴリズムを対象としているため，問題の持つ並列性をハードウェアに反映させて処理することが容易でない．②は，問題の持つ並列性を可能な限り表現することが可能であり，良く設計された並列アルゴリズムを用いればプログラムの効率的な実行が期待できる．一方，並列言語に対してプログラミングの困難さがよく言われる．しかし，偏微分方程式や離散事象シミュレーションのように並列性が自明なものも多く，また並列アルゴリズムに基づいた記述を行う限りにおいてプログラミングは必ずしも困難な作業ではない．このようにどちらにも長所，短所があり，互いの欠点を補いあったプログラミング環境が望まれる．そこで我々は，上記の２つの手法に基づいた統合的なプログラミング環境の開発を進めている[6]．以下では，②の並列プログラミング言語について述べる．

これまで数多くの並列プログラミング言語が提案されてきた[7]．しかし，Ada[8] をはじめとしてそのほとんどは分散処理言語とも称されるシステム記述用言語であり，大規模なマルチプロセッサ上で並列処理応用問題を記述するための十分な考慮はなされていない．Occam[9] は，マルチプロセッサ上に実装された並列プログラミング言語であるけれども，対象とするマルチプロセッサは強く限定されている．

そこで我々は，大規模なマルチプロセッサシステムを対象とし，並列応用問題を記述するための並列プログラミング言語SERVEの設計開発を進めている．本稿では，SERVEの概要，応用例および処理系について述べる．

2. SERVEの設計方針

並列プログラミング言語SERVEは以下の事項を設計方針とした．

（1）並列性の指定方法

並列性表現のレベル（プロセス）には，Occamのように文レベルで指定するものや，Adaあるいは並列オブジェクト指向言語のようにひとまとまりの処理レベルで指定するものがある．また別の形態として，関数型言語における関数や論理型言語におけるゴールのように，細かいレベルで並列性を表現するものもある．我々が対象としているマルチプロセッサ構成の並列計算機との整合性を考慮した場合，並列実行の単位である粒度は比較的粗い方が効率的な実行が期待できる．また，ユーザに対する記述性を考慮した場合も，機能単位ごとにまとめて並列性を指定する方が容易である．したがって，並列性は局所データとそれに対する一連の逐次操作を定義したものを単位として明示的に指定する．SERVEでは，これをプロセスとする．

（2）プロセス間通信の方法

プロセス間通信の方法は以下の２つの方法に大別される．

① 共有変数による通信

② メッセージによる通信

さらに２つの方法を合わせ持つ場合もある．①の方法の利点は効率的な通信が可能なことである．しかし，排他制御のための記述やプロセス間の相互作用がわかりにくいという問題があり，プログラムが複雑になると予期しない誤りを犯す可能性が高くなる．一方②の方法では，通信と同期操作が一体化して行なわれ，構造化よる理解しやすいプログラムの記述が可能である．したがって，プロセス間通信は②の方法に基づいて行う．また，問題に応じて柔軟な記述ができるように，多様な通信方法を提供する．

（3）高並列性の記述

大規模なマルチプロセッサに対する高並列なプログラムを容易に記述することが目的である．そのためには，解くべき問題をできるだけ多くの並列実行可能な処理に分割することが必要である．この分割には，一般に以下の２つの方法がある．

① 負荷分割，すなわち均質処理への分割

② 機能分割，すなわち不均質処理への分割

大規模マルチプロセッサにおけるプログラミングを考えた場合，①の方法のほうが容易である．なぜなら，機能の異なる多数のプロセスを記述することは容易で

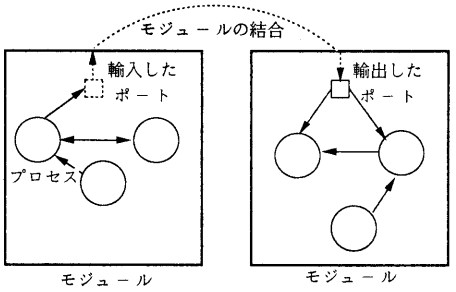


図1 SERVEプログラムの構造

はないからである．①の方法は，さらに以下の2つの方法に分類される．

① データに基づく分割

② 問題の細分化による分割

①の例としては，偏微分方程式を離散化して解く場合や行列計算などがある．②の例としては，分割統治法により解かれる問題や探索問題がある．これらの処理を記述するための言語機能として，まず①に対しては多数の均質プロセスを同時に生成しそれらが異なるデータを扱えることが，②に対してはプロセスを再帰的に生成することが必要である．SERVEはこれらの要件を満足し，高並列性を容易に記述できるようにする．

3. SERVEの言語機能

並列プログラミング言語SERVEの提供する言語機能について述べる．SERVEのプロセス内部の逐次的な部分は，現在広く普及しているC言語の構文に基づいて行なわれる．

3.1 プログラム構造

SERVEのプログラムは図1に示すように，1つ以上のモジュールから構成され，さらにそのモジュールは複数の並列プロセスから構成される．

規模の大きなプログラムの作成には，機能単位に分割してコーディング，コンパイル，テストできることが必要である．モジュールはその単位になるものである．さらに汎用的なモジュールは，他のプログラムを構築する時に再利用される．モジュールの結合は，関数やメッセージのためのバッファ（ポートおよびエントリ）を輸出入することで行う．

3.2 プロセス

解くべき問題によっては並列度が静的に定まるものと処理の過程において変化するものがある．そこでSERVEでは，並列実行の単位であるプロセスの生成は，次の2つの方法によって行なわれる．

① 静的生成：プログラムの開始時に暗黙的に生成される．

② 動的生成：プログラムの実行中に明示的な文により生成される．

静的プロセスは，以下ように宣言することによって生成される．なお，動的生成を行なう場合に必要なプロセスタイプの宣言も同様の形式で行なわれる．

```
process プロセス名
  ポート，エントリの宣言
  {内部動作の定義}
```

静的なプロセスは，それを識別するための名前と内部の動作によって定義される．また，必要ならばそのプロセスが専有して使用するメッセージ・バッファ（ポートまたはエントリ）が与えられる．

動的なプロセスの生成は，前もって宣言されたプロセスタイプ名を指定してcreate文を用いることによって行なわれる．

```
x＝create（a）
```

create文は，プロセスタイプaのプロセスを新たに生成しその値（プロセス値）を返す．xはプロセスタイプaの変数であり，プロセス値を保持する．新たに生成されたプロセスは，この変数により識別される．

さらに静的生成では，多数の均質プロセスを表現するためにインデックス付きのプロセス配列を宣言することが可能である．

```
process a［I：N］
```

この場合，同一処理構造を持つN個のプロセスが生成される．また，インデックスIは各プロセスに固有な0からN－1までの値をとる．このインデックスには以下の2つの応用がある．

① 問題のデータ空間の分割

プロセス配列の各プロセスは，インデックスの値に基づいてデータ空間の異なる領域を処理する．

② プロセス配列における各プロセスの識別

プロセス配列中の各プロセスの間で互いに通信しあう場合に使用する．

以上の方法で生成されたプロセスは，静的／動的生成の如何にかかわらず自分自身の処理を終えた時点で消滅する．

3.3 メッセージ通信

（1）通信プリミティブ

SERVEはメッセージの通信プリミティブとして次のの2つを用意している．

① 非同期式通信（send／receive）

非同期式通信は単方向通信であり，メッセージを一方的に送りつける場合に使用する．非同期式通信のためのメッセージ・バッファをポートと呼ぶ．送信プロセスはメッセージ送信後直ちに実行を再開する．

- send ポート名（式リスト） ［to 送り先］
- receive ポート名（変数リスト）［from 送り元］

② 同期式通信（call／accept）

同期式通信は双方向通信であり，クライアント・サーバ関係すなわち手続き呼出し的な通信を行う場合に使用する．同期式通信のためのメッセージ・バッファをエントリと呼ぶ．呼出しプロセスは返値メッセージが返されるまで実行が封鎖される．

- call エントリ名（式リスト） ［to 呼出し先］
  wait（返値変数リスト）
- accept エントリ名（変数リスト）
  ［from 呼出し元 ］ 〔複合文〕

返値メッセージはaccept複合文中のreply文により返される．

reply（返値リスト）

また，reply文の実行以後は，accept文を抜けてその後の処理を継続する．

以上に示したように，SERVEのメッセージ通信は，通信相手の指定に関して対称形である．すなわち，相手プロセスを指定してもしなくてもよい．sendおよびcall文において通信相手が省略できるのは，後述するように，グローバルに宣言されたポートあるいはエントリに対してのみである．この場合，任意の受信プロセスを対象とすることができる．また，receiveおよびaccept文において相手プロセスが省略された場合は，任意のプロセスからの受信が可能となる．通信相手の指定は，プロセス名やプロセス変数などによって行なわれる．

（2）ポート／エントリの宣言

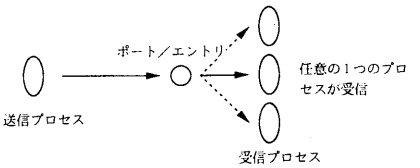
ポート／エントリの宣言はその位置により以下の2つの方法が可能である．

① プロセス内部でのローカルな宣言

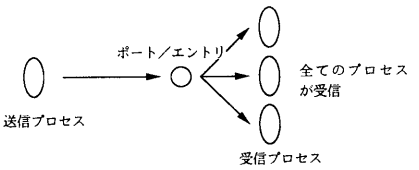
② プロセス外部でのグローバルな宣言

①の場合，宣言を行なったプロセスのみが専有して受信操作を行う．一方②の場合は，ポート／エントリに対して任意のプロセスが受信操作可能となる．すなわち，宣言位置によって，受信操作に関するスコープ・ルールが適用される．

ポートおよびエントリの宣言では，転送されるパラメータの型が宣言される．



(a) ANY型対多通信



(b) ALL型対多通信

図2 グローバルなポート／エントリによる対多通信

port ポート名（型リスト）

entry エントリ名（型リスト）:（型リスト）

エントリの2番目の型リストは返値の型である．パラメータの型としては，C言語で用意されているポインタ型以外の通常のデータ型のほかに，配列データを送受信するための型が宣言できる．

port p（int［N］）

この場合，ポートpを通して整数型でサイズNの配列データが送受信されることを意味する．

（3）対多メッセージ通信

グローバルに宣言されたポート／エントリを用いることにより，以下の2種類の対多通信が可能である．

① ANY型対多通信（非決定的送信）（図2（a））

受信プロセスを特定しない通信形態である．実際の受信プロセスは，そのポート／エントリに最初に受信操作を行なったプロセスが選択される．この通信形態は，通信相手の指定を省略するか，あるいはANYと指定することで行なわれる．

② ALL型対多通信（ブロードキャスト）（図2（b））

その時点で存在する自分以外の全てのプロセスに対して同一データを送信する．この通信形態では，通信相手をALLと指定する．

さらに，対多通信の別の形態として，プロセス配列に対するマルチキャスト通信を指定することが可能である．これは,通信相手としてプロセス配列の部分範囲（例えばp［k..n］）を指定する．この場合，ポート／エントリはグローバルである必要はなく，各プロセスの個別な

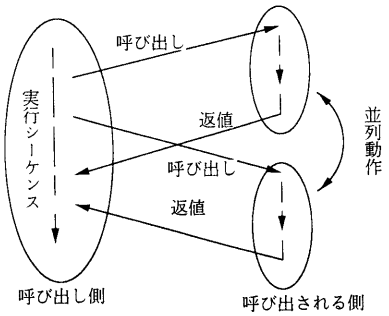


図3 async 文による並列呼び出し

ポート／エントリに送信される．

同期式通信を用いたブロードキャスト通信およびマルチキャスト通信では，最初に返され返値のみが有効であり残りは捨てられる．

（4）並列呼出し

同期式メッセージ通信は，手続き呼出し的な通信が可能であるが，呼出しを行なったプロセスは返値が返されるまで実行が封鎖される．したがって，複数のプロセスを並列に呼出したい場合でも，通常の同期式メッセージ通信では逐次的な呼び出ししか行なえない．しかし，これでは並列呼び出しを行ないたいような問題，例えば分割統治法によって解かれるような問題に対する高並列な動作の記述ができない．このような動作を記述するために，async文を用意した．async文はその内部にcall文を列挙したものである．

async {call文の並び}

async文の動作の意味は，列挙されたすべてのcall文の呼びC出しを返値メッセージの到着を待たずに行い，その後で返値メッセージの受取りを行なうことである．これにより，async文中で呼び出された全てのプロセスは並列動作することになる（図3）．

3.4 大域変数

SERVEではプロセス間の共有変数は許していない．したがって，配列のような大規模な初期データを各プロセッサで負荷分散させる場合には，その部分領域の処理を担当する各プロセスへメッセージ通信を用いてデータを分配しなければならない．これは，余分な記述が必要になるばかりか，プログラムの実質的な処理の開始を遅らすことになる．そこで，リードオンリの大域変数を許すことにした．各プロセスはデータを大域変数から自分の局所変数へ読み込んで処理を行う．つまり，初期データの分割はその処理を担当する各プロセスが自律的に行う．

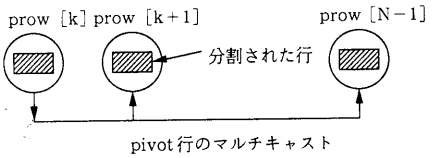


図4 ガウスの消去法の処理構造

この大域変数はプロセスmainによって初期設定される．プロセスmainは，最初に実行を開始するプロセスであり，その他の通常のプロセスはプロセスmainが終了した後に起動される．プロセスmainは，通信を行なわず大域変数への書き込みが許される以外は通常のプロセスと同じである．

3.5 プログラムの終了

並列プログラムの終了は，逐次プログラムのそれに比べ重要な概念である．SERVEプログラムが終了するのは，すべてのプロセスが終了するか，または，生存している全てのプロセスが，非決定的受信を表現するselect文の終了分岐で待っているときである．ここでSERVEのselect文は，Adaのselect文の部分的な機能を有する．SERVEのプロセスは，終了に関して親子関係はなく，自分が生成したプロセスの終了を待たずに終了する．

4．応用例と考察

4.1 応用例

代表的な応用例をSERVEで記述し，SERVEの記述力を検討する．

（1）ガウスの消去法

これは，連立一次方程式Ax＝bの代表的解法の1つである．ここでは，N×N係数行列Aに消去法を行う部分について述べる．

まず，行列AをN個の行に分割し，それぞれに独立な均質プロセスを割り付ける．これは，プロセス配列prow［I：N］（Iは0からN−1の値を持つ）を生成して行う．行列Aは（プロセスmainが初期化を行う）大域変数として宣言され，各プロセスはインデックスIの値に基づいて分割を行う．すなわち，プロセスprow［i］は第i行を保持する．その後各プロセスは，N−1回の繰り返し計算を行う．繰り返し回数kがインデックスIと等しい場合，そのプロセスはpivot行を保持していることを

意味する．その場合，図4に示すようにpivot行を他のprow［k＋1］からprow［N－1］のプロセスにマルチキャストする．その後，各プロセスは自分が保持している行に対して消去法を適用する．以上の処理を，SERVEで記述したプログラムの主要部分を図5に示す．

```
double a [N] [N];

process prow [I : N]
port p (double [N]);
{
    double myrow [N], pivot [N];
    int k;

    for (k=0 ; k<N ; K++)
        myrow [k] =a [I] [k];

    for (k=0 ; k<N-1 ; k++)
        if (I >= k) {
            if (I == k) send p (myrow)
                    to prow [k+1..N-1];
            else        receive p (pivot);
            elimination (I,k,myrow,pivot);
        }
}
```

図5 ガウス消去法のSERVEプログラムの主要部

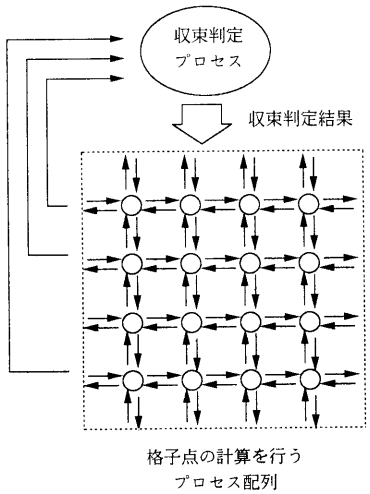


図6 ラプラス方程式の処理構造

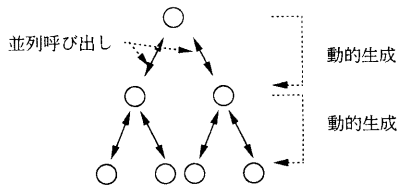


図7 マージ・ソートの処理構造

（2）ラプラス方程式

ラプラス方程式を均一正方形によりに離散化して解く場合，各格子点の近似式は次のように与えられる．

$$U_{i,j}(k+1) = (U_{i+1,j}(k) + U_{i-1,j}(k) + U_{i,j+1}(k) + U_{i,j-1}(k)) / 4$$

ここで，$U_{i,j}(k)$は格子点（i，j），反復回数kにおける値である．この反復計算は，すべての格子点で

$$| U_{i,j}(k+1) - U_{i,j}(k) | < \varepsilon$$

となった時点で終了する．

この処理を行うために，図6に示すように2種類のプロセスを用意する．1つは，各格子点の処理を行う2次元のインデックス付きプロセス配列であり，もう1つは，計算の収束判定を行うプロセスである．各格子点では，上下左右の格子点に対しデータを送り，それらの格子点からデータを受け取って平均するという上式の計算を反復して行う．この場合，上下左右の格子点（プロセス）との通信および境界点の判定は，プロセス配列のインデックスの値を評価して行なわれる．また，収束判定プロセスは，各格子点から個別な収束判定結果を受け取り，全体的な収束判結果をプロセス配列に返す．この時，全体的な収束判定結果の送信はブロードキャスト通信を用いて行なわれるため，そのとき使用するポートはグローバルに宣言しておく．

（3）マージ・ソート

これは，分割統治法によるソーティングの1つである．すなわち，与えられたデータ列を2つの部分列に再帰的に分割していき，それぞれを独立にソートして得られたデータ列を併合するという手法である．

このためには，図7に示すように分割の度に2つの子プロセスを動的に生成する．さらに，それぞれの子プロ

セスを，同期式通信にasync文を適用して独立に呼び出す．以上の処理をSERVEで記述したプログラムの主要部分を図8に示す．ただし，便宜上与えられるデータ列の長さを2k（k≧1）としている．

```
process type sort
entry e (int,double [N]) : (double [N]) ;
{
procvar quick left,right ;
int length ;
double data [N],l_data [N],r_data [N] ;

 accept e (length,data)  {
   if (legth==2)  {
       if (data [0]  >  data [1])
           swap (&data [0],&data [1]) ;
   }
   else  {
       divide (lenth,data,l_data,r_data) ;
       left    =  create (sort) ;
       right   =  create (sort) ;
       async  {
           call e (length/2,l_data)
                    to left wait (l_data) ;
           call e (length/2,r_data)
                   to right wait (r_data) ;
       }
       merge (length,data,l_data,r_data) ;
   }
   reply (data) ;
 } }
```

図8 マージ・ソートのSERVEプログラムの主要部

（4）巡回セールスマン問題

これは，グラフ上の頂点をすべて通りその間のコストが最小となるような経路を求める問題であり，探索問題の1つである．このような問題には，一般的に分枝限定法が適用される．分枝限定法は，最適解の推定値に基づいて探索木の枝刈りを行ないながら，効率的に探索を行う手法である．

この処理を行うために，探索木に従って動的にプロセスを生成していくことも考えられるが，ANY型対多通信の応用例を示すために固定数（例えば，プロセッサ台数分）のプロセスにより解く方法を示す．

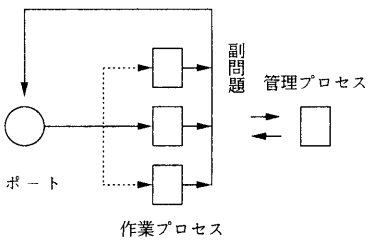


図9 巡回セールスマン問題の処理構造

ここでは，図9に示すように2種類のプロセスを用意する．1つは，最小コストの推定値とそのときの経路を保持している管理プロセスであり，もう1つは，実際の探索を行う作業プロセス（プロセス配列）である．作業プロセスは，グローバルなポートから副問題（途中までの経路とその間のコストおよび次に探索すべき頂点）を受信してきては評価を行ない，さらにいくつかの副問題に分割してグローバルなポートに送信する．この場合，グローバルなポートへ通信相手を特定せずに送信（ANY型対多通信）すれば，その副問題の処理は手の空いた作業プロセスが受信して行うことになる．実際の副問題の評価において，管理プロセスから最小コストの推定値を受信してきて枝刈りを行う．また，作業プロセスは一巡経路が見つかった場合，あるいは，枝刈りを行なった場合に，管理プロセスに結果メッセージとして送信する．管理プロセスは，作業プロセスからの結果メッセージの受信と最小コストの推定値の要求メッセージの受信を，select文中で待つ．この問題で使用するグラフの構造は大域変数としておけば，各プロセスが独自に持つ必要はない．

4.2 考察

ここでは，4.1の応用例からSERVEについて簡単に考察する．

SERVEのプロセス生成については，インデックス付きプロセス配列，および動的なプロセス生成，また，メッセージ通信については，ANY／ALL型通信，マルチキャスト機能など，多様な機能を提供しているので，問題に応じて比較的素直に記述できる．マージ・ソートなどの問題には，プロセスを再帰的に生成させることによってアルゴリズムを素直に記述できる．このとき，並列呼出し機構が有効となる．並列呼び出し機構は，非

同期式通信を用いて記述することもできるが，記述が複雑になる，また，ラプラス方程式の離散化近似解法において,上下左右の隣接プロセスへのデータの送信は，マルチキャストで行なうことが考えられる．しかし，SERVEで提供しているマルチキャスト機構は連続するプロセス配列にのみしか適用できない．記述性については，今後大規模プログラムをSERVEで記述することによってさらに検討し，言語仕様の整備を行なっていく必要がある．一方，SERVEはメッセージ指向の言語であり，また，多様な機能を提供しているので,通信処理が重たくなるという問題点がある．また，巡回セールスマン問題に適用した手法では，ポートのバッファ長が有限な場合，デッドロックを起こす可能性が生じる．この解決策の1つとして，バッファを動的に生成して論理的には無限長とする方法が考えられるが，その場合さらに通信処理が重たくなる．これについては今後の検討課題である．

5．処理系

5.1 SERVEプロセスの実現方法

SERVEプロセスを並列動作させる方法には，仮想アドレス空間を割当てる単位に着眼して，次の2つが考えられる．

① 各SERVEプロセスに個別の仮想アドレス空間を与える．

② すべてのSERVEプロセスに同一の仮想アドレス空間を共有させる（いわゆるスレッドとして実現する）．

これら2つの方法を次の項目について比較検討する．

（1）効率

（a）プロセス生成： 方法①では，ファイル・システムからの実行形式ファイルのロード，およびOSの各種管理表の確保と設定を新たに行なう必要がある．一方，方法②ではスタック領域，およびコンテキストを格納する領域の確保と設定など，方法①の一部でよい．したがって，方法②の方がプロセス生成に要する時間が短かくてすむ．

（b）コンテキスト切り換え： 方法①では仮想アドレス空間の切り換えをともなうので，コンテキスト切り換えが方法②よりも重たくなる．

（2）消費される計算機資源

（a）OSの資源： 方法①では，SERVEプロセス数だけプロセス・コントロール・ブロック，アドレス変換表などが必要となる．一方，方法②ではSERVEプロセスのコンテキストを保存する領域は新たに必要となるが，プロセス・コントロール・ブロック，アドレス変換表などは1つでよい．したがって，方法②の方が必要とされるOSの資源量が少なくてすむ．

（b）記憶領域： ページング・システムを仮定する．方法①では，1つのSERVEプロセスに対して，コード，データ，スタックをそれぞれ少なくとも1ページづつ割当てる必要がある．したがって内部断片化が生じやすくなる．一方，方法②では，コード，データ領域の割当ては全SERVEプロセスのコード，データについてまとめて行なえばよいから，内部断片化が方法①よりも生じにくい．したがって，必要とする記憶領域は方法②の方が少なくてすむ．また，必要となるページ数が少なくなることから，ページフォールト頻度も方法①よりも少なくなると考えられる．

以上見てきたように，種々の点から方法①よりも②が優れているので，SERVEプロセスをいわゆるスレッドとして実現する．

5.2 擬似並列用SERVE処理系

現在，“可変構造型並列計算機”およびその上の並列OSは開発中であるので，シングルプロッセサ（Sun－4）上で擬似的に並列動作させるための処理系の開発を行なっている．処理系は，C言語を中間言語とするトランスレータであり，既存のCコンパイラを流用する．また，SERVEの各プロセスは，5．1の実行方式に基づき，Sun－4のLightweight Processとして実現する．我々が開発中の並列OSは，並列処理モデルとしてタスク／スレッドモデルを提供しており,また，C言語レベルでスレッドを制御することが可能である．したがって，“可変構造型並列計算機”上で実並列処理を行なう場合にも，SERVEの処理系はC言語へのトランスレータとして構成できる．

6．おわりに

大規模マルチプロセッサで実行されることを意図した並列処理問題記述用言語SERVE言語機能,簡単な応用例および処理系について述べた．SERVEは，言語機能として多様なプロセス生成法とメッセージ通信機能を提供している．また，4つの応用例にSERVEを適用することで，その記述性を検討した．現在，処理系の開発を鋭意進めている．

謝辞

現在我々とともに“可変構造型並列計算機”のハードウエア，OSの設計・開発に携わっている森，蒲池，福澤，岩田，甲斐，草野，恒富，上野，杉山の各氏，および，

日頃ご討論いただく富田研究室の皆様に感謝致します.

参考文献

[1] K.Murakami et al.: "The Kyushu University Reconfigurable Parallel Processor－Design Philosophy and Architecture－, "Proc. IFIP 11th World Computer Congress, pp.995-1000 (Sept. 1989).

[2] K.Murakami et al.: "The Kyushu University Reconfigurable Parallel Processor－Design of Memory and Intercommunication Architectures －, "Proc. 1989 Int'l. Conf. Supercomputing, pp.351-360 (June 1989).

[3] 森ほか："可変構造型並列計算機のPE間メッセージ通信機構," 情報処理学会論文誌, vol.30, no.12, pp.1593-1602 (1989年12月).

[4] 福田ほか："可変構造型並列計算機の並列／分散オペレーティング・システム," 情報処理学会オペレーティング・システム研究会資料, 89-OS-43-8 (1989年6月).

[5] 福澤ほか："可変構造型並列計算機の並列／分散オペレーティング・システム－スレッドの実現－," 情報処理学会オペレーティング・システム研究会資料, 89-OS-45-6 (1989年11月).

[6] 村上ほか："統合型並列化コンパイラ・システム," 情報処理学会第40回全国大会投稿中.

[7] G.R.Andrewa and F.B.Schneider: "Concepts and Notations for Concurrent Programming," ACM Computing Surveys, Vol.15, No.1, pp.3-43 (1983).

[8] J.R.Barnes: Programming in Ada, Addison－Wesle (1984).(牛島, 荒木: プログラミング言語Ada, サイエンス社).

[9] 尾内理紀夫：Occamとトランスピュータ, 共立出版 (1986).