

## 解説



## マルチプロセッサスーパーコンピュータ PHI の研究開発

マルチプロセッサスーパーコンピュータ用  
並列記述言語†

山本 富士男†† 出本 学†† 梅谷 征雄††

## 1. はじめに

技術計算の各分野においては、ベクトル型スーパーコンピュータを用いた数値シミュレーションは実用期にあるが、一方では、解析すべき対象の一層の拡大と解析の精密化にともなう計算の大規模化の必要性から、より高いコストパフォーマンスで高速計算が可能と期待される並列型スーパーコンピュータへの関心が高まっている。

しかしながら、現時点では、このような並列マシン上で一つの計算を多数のプロセッサに分配して効率良く計算させるための仕事は、多くの場合ユーザの人手作業に委ねられていると言っても過言ではない。すなわち、ユーザは、自分の問題の解法を使い慣れた FORTRAN や C 言語で記述するだけでは済まず、計算手順と主要なデータを、対象並列計算機に適合するように分割して表現し、さらにその分割に対応して必要となる相互通信を明示するという仕事を追加しなければならない。アーキテクチャ特性を考慮しなければならない度合いは対象計算機システムによって異なるが、一般に分割に関しては明確な指針もないうえ、複雑で間違いやすい仕事となる。これが、並列プログラミングを単一計算機の場合よりも困難にしている要因の一つである。

ユーザをこのような負担から解放しないかぎり、真に実用的な並列処理はありえない。これを解決するためのソフトウェア技術の課題は多岐にわたるが、ここでは高水準言語によるアプローチをとる。その狙いは、ユーザが種々の並列アーキテクチャの細部にとらわれずに、論理的なレベル

で計算アルゴリズムの並列化を促進できるような手段を与えることにある。さらに、このような論理レベルで明示された並列性を基に、処理ソフトウェアによって自動的にプログラムを分割し並列アーキテクチャへマッピングすることである。

本解説では、このような目的をもって、通商産業省工業技術院大型プロジェクト「科学技術用高速計算システムの研究開発」の一環として開発した並列記述言語 PARAGRAM について述べる。

2. では本言語の特徴を簡単に述べ、3. では、具体的な数値計算の例を取り上げ、それが本言語で明確に記述できることを示す。4. では言語処理ソフトウェアの概要、5. では、例題に対する記述性と言語処理ソフトウェアによる翻訳結果の実行性能について検討する。

## 2. 言語仕様の概要と特徴

並列処理記述言語 PARAGRAM は、共有記憶モデルに基づき、数値シミュレーションプログラムにおける演算並列性を明確に表現できることを第一の目的とする。本言語では、汎用性を重視するが、一方ではこの分野で多用される疎行列操作に関して他言語にない特徴機能を提供するよう努めた。また、従来の FORTRAN や PASCAL などの手続き型言語との融和を図っている。言語仕様の詳細は文献 1) にあるので、ここでは特徴の一つである配列型・手続き型並列性記述機能について簡単に述べる。

まず、配列型並列性を簡潔に記述するため、行列要素が帯状、および不規則な位置に疎に配置されている場合、また、行列要素がブロック（部分行列）で階層的に構成されている場合のデータの定義機能をもっている。これらの行列同士の演算、たとえば行列  $A, B$  の加算は疎構造の種類によらず、 $A = A + B$  のように記述できるようにし

† A Parallel Programming Language by Fujio YAMAMOTO, Manabu DEMOTO and Yukio UMETANI (Central Research Laboratory, Hitachi Ltd.).

†† (株)日立製作所中央研究所

た。疎構造は、帯行列では構造パラメータ *band* と帯幅、一般疎行列では構造パラメータ *sparse* とインデックスアレイによって宣言される。インデックスアレイは疎行列の非零要素の位置を指定するためのものである。(図-1の下段参照) 行列演算後の非零パターンが変化する場合、インデックスアレイは自動的に更新される。このように、PARAGRAM では明快な疎構造操作を目指す。

一方、手続き型並列記述機能としては、同種手続き間の並列実行を指示する *pcalc for* ループと異種手続き間の並列実行を指示する *pcalc case* の2形式を備えている。(図-1の上段参照) 多様な並列構造を表現できるよう、これらは相互に何重にもネスト可能とし、並列実行手続き間のデータフロー待ち合わせ (*send/receive*) 機能も用意した。並列実行手続きの記述では配列レベルの演算が使えるので、手続き型・配列型両並列性の混在が可能となっている。以上の機能について、FORTRAN 言語との比較を図-1の右欄に示したので、PARAGRAM 記述の簡潔性をうかがうことができると思われる。

### 3. 帯行列解法プログラムの記述

この章では上記に述べた PARAGRAM 言語の特徴のうち、疎行列構造の階層的定義機能が数値計算プログラムにおいてどのように適用できるかを具体例に沿って説明する。例題として、構造解析、流体解析などの偏微分方程式の離散化の結果として得られる、帯行列係数の連立一次方程式の並列解法を取り上げる。

#### 3.1 帯行列の分割

問題は連立一次方程式  $AX=Y$  を解くことである。係数行列  $A$  は非対称であり、サイズが  $N \times N$ 、帯幅  $2m+1$  の帯行列である。簡単のため  $N=2^d$  と仮定する。大きな方針として、行列  $A$  と解ベクトル  $X$ 、および右辺ベクトル  $Y$  を  $p$  個 ( $p=2^f$ ,  $f$  は正の整数) のブロックに分割し、大局的にはこれらのブロックに関して並列に求解する

ブロック並列記述機能

| 並列構造       | PARAGRAM                                                                    | FORTRAN                                                                                                                |
|------------|-----------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| 同型手続き間<br> | <pre>pcalc for I in [1:N];   P(I); end pcalc;</pre>                         | <pre>do I=1,N   call % proc('P', I) repeat do I=1,N   call % wait('EOP', I) repeat</pre>                               |
| 異手続き間<br>  | <pre>pcalc case:   case(1): P;   case(2): Q;   case(3): R; end pcalc;</pre> | <pre>call % proc('P') call % proc('Q') call % proc('R') call % wait('EOP') call % wait('EOQ') call % wait('EOR')</pre> |

疎行列記述機能

| 行列構造       | PARAGRAM                                                                     | FORTRAN                                                                                                                            |
|------------|------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| (帯行列)<br>  | <pre>array(A,B) size([1:N],[1:N]) structure(BAND(M)); A=A+B;</pre>           | <pre>real A(N,2*M+1), B(N,2*M+1) do I=1,K   do J=max(1,M+1-I,     min(2*M+1,N+M-I))     A(I,J)=A(I,J)+B(I,J)   repeat repeat</pre> |
| (一般行列)<br> | <pre>array(A,B) size([1:N],[1:N]) structure(SPARSE(array(IX))); A=A+B;</pre> | <pre>real A(NN), B(NN) integer IX(NN) do I=1,NZ   A(IX(I))=A(IX(I))   +B(IX(I)) repeat</pre>                                       |

図-1 PARAGRAM 言語の特徴

$$AS[i] = \begin{bmatrix} C_i & A_i & B_i \\ q-m & m & q-m & m & q-m & m \\ \begin{pmatrix} O & G_{11} \\ O & G_{12} \end{pmatrix} & \begin{pmatrix} E_{11} & E_{12} \\ E_{11} & E_{12} \end{pmatrix} & \begin{pmatrix} O & O \\ F_{11} & F_{12} \end{pmatrix} \end{bmatrix} \begin{matrix} q-m \\ m \end{matrix}$$

$$X_i = \begin{pmatrix} U_{i1} \\ U_{i2} \end{pmatrix} \begin{matrix} q-m \\ m \end{matrix} \quad Y_i = \begin{pmatrix} H_{i1} \\ H_{i2} \end{pmatrix} \begin{matrix} q-m \\ m \end{matrix}$$

図-2 帯行列のブロック分割

ことを考える。すなわち、 $A$  を  $q \times q$  ( $q=N/p$ ) のサイズブロック  $C_i, A_i, B_i$  ( $i=1, 2, \dots, p$ ) からなる3重ブロック対角行列にする。(図-2参照)  $A_i$  が対角ブロック、その左右に  $C_i$  と  $B_i$  が置かれる。また、未知数ベクトル  $X$  を  $X_i$  に、右辺ベクトル  $Y$  を  $Y_i$  に分割する。さらにこれらのブロックを図-2のように細分する。そして新たに、 $C_i, A_i, B_i$  からなるブロック  $AS[i]$  を構成する。ただし、 $G_{11}, G_{12}, F_{11}, F_{12}$  は零行列とみなす。

#### 3.2 並列アルゴリズム

上記のように分割された帯行列に対する基本アルゴリズムを以下に示す。

## (1) フェーズ 1

- ステップ 1:

$E_{i1}$  を LU 分解し,  $E_{i1}$  を単位行列化する。これは  $i=1, 2, \dots, p$  について並列に実行できる。

- ステップ 2:

$E_{i3}$  を消去する。これは  $i=1, 2, \dots, p$  について並列に実行できる。

- ステップ 3:

$F_{i1}$  を消去する。これは  $i=1, 2, \dots, p-1$  に関して並列に実行できる。ただし、ステップ 2 が完

了している必要がある。

以上の処理で,  $U_{i1}$  に関する方程式を分離し,  $U_{i2}$  に関する方程式だけに着目できる。すなわち解くべき行列のサイズが半分になったとみなすことができる。

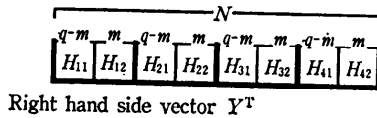
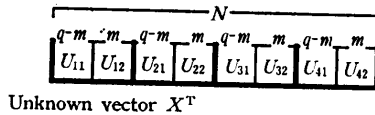
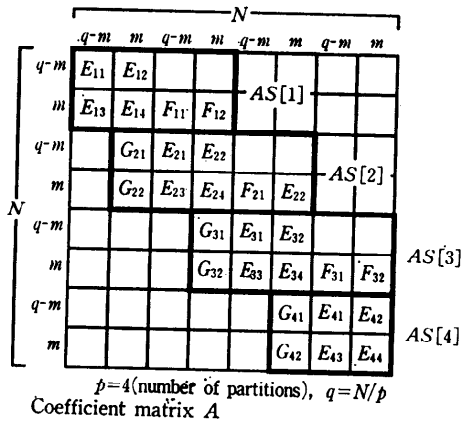
## (2) フェーズ 2

縮約された行列の方程式を解き  $U_{i2} \{i=1, 2, \dots, p\}$  を求める。

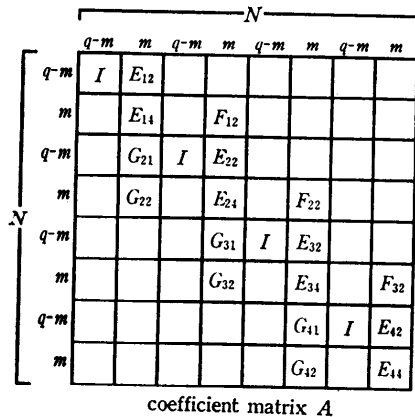
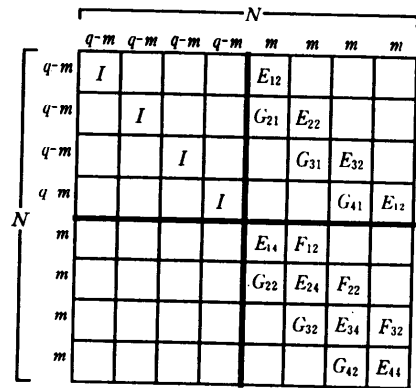
## (3) フェーズ 3

上記で求められた  $U_{i2}$  から  $U_{i1} \{i=1, 2, \dots, p\}$  を求める。これは後退代入であり, フェーズ 2 が完了していれば  $i$  に関して並列に実行できる。■

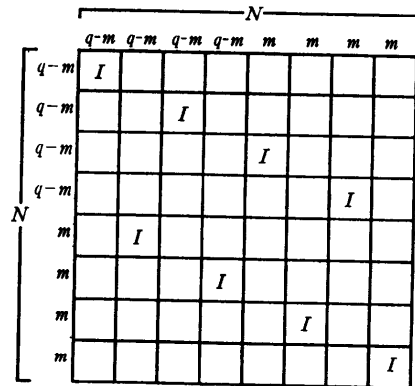
上記基本アルゴリズムは S. L. Jhonsson<sup>2)</sup> に基づいたものであるが, ここではフェーズ 2 開始時点での縮約された行列に再びフェーズ 1 を適用し, 行列サイズをさらに順次半減させてゆく方式<sup>3)</sup>を取っている。具体例を以下に示す。図-3 は  $p=4$  の場合の例である。フェーズ 1 の終了後の



(a) 帯行列のブロック分割例



(b) ステップ 3 のフェーズ 1 後の係数行列



(d) Matrix  $R^T$

係数行列は図-3 (b) のようになる。これに図-3 (d) の行列  $R$  と  $R^T$  を  $A$  の両側に掛けると図-3 (c) の縮約された行列が得られる。フェーズ 2 はこの結果の  $U_{i2}$  を解くフェーズである。フェーズ 1 の縮約行列の東南 1/4 の部分はやはりブロック帯行列であり、この部分をふたたびフェーズ 1 の処理をくり返すことでさらに行列を縮約する。これを、残りブロックが一つになるまで階層的にくり返す。ただし、ブロック内のサブブロックの行列サイズは、第一回目のフェーズ 1 では  $(q-m) \times m$  または  $m \times (q-m)$  であったものも全て  $m \times m$  のサイズとなる。最後に残った唯一つのブロックは LU 分解で求解する。

フェーズ 3 の後退代入による求解では、フェーズ 1 を階層的に適用することに対応して、後退代入も階層的にくり返す。階層間のくり返しは逐次的に実行しなければならないが、一つの階層内での後退代入はそのとき解くべき未知数の個数分の並列度がある。

### 3.3 PARAGRAM による並列アルゴリズムの記述

上記の解法アルゴリズム全体を、本言語によって完全に記述した。プログラム全体は 701 行となった。このうちの一部を図-4 に示す。図-4 の 1 行目から 22 行目までは係数  $A$  の構造定義である。ここではデータの階層的記述を使っている。上記で説明した行列の構造、すなわち、行列  $A$  は部分行列(上記ではブロックと呼んだ)  $AS[i]$  から構成され、各  $AS[i]$  の左上角の位置、すなわち  $A$  の中の相対位置は 4 行目の  $orig$  パラメータによって指定されている。さらに  $AS[i]$  は部分行列  $G1, G2, E1, \dots$  などに分割される。この記述は上記でアルゴリズムを考えたときのデータ構造を素直に表現しており、分かりやすいものとなっている。23 行から 31 行は未知数ベクトル  $X$  の構造であるが、行列  $A$  の場合と同様に記述することができる。

一方、処理手続きに関しては、80 行目から 88 行目は上記フェーズ 1 のステップ 1 の処理である、

```

1 array
2 Ar*4 size([1:NDM1],[1:NDM2])
3 structure(blocks(AS[[1:IP]])),
4 (AS[i] size(JQ,[1:2*IQ+IM]) subarray of A(orig(IQ*(i-1)+
      IQ*(i-1)+1))
5     structure(blocks(G1,G2,E3,E4,F1,F2,E1,E2)),
6     G1 size(JQM,JM) subarray of AS(orig(1,1))
7     structure(dense),
8     G2 size(JM,JM) subarray of AS(orig(IQM+1,1))
9     structure(dense),
10    E1 size(JQM,JQM) subarray of AS(orig(1,IM+1))
11    structure(dense),
12    E2 size(JQM,JM) subarray of AS(orig(1,IQ+1))
13    structure(dense),
14    E3 size(JM,JQM) subarray of AS(orig(IQM+1,IM+1))
15    structure(dense),
16    E4 size(JM,JM) subarray of AS(orig(IQM+1,IQ+1))
17    structure(dense),
18    F1 size(JM,JQM) subarray of AS(orig(IQM+1,
      IQ+IM+1))
19    structure(dense),
20    F2 size(JM,JM) subarray of AS(orig(IQM+1,2*IQ+1))
21    structure(dense)
22 ) for i in [1:IP],
23 Xr*4 size([1:NDM1])
24 structure(blocks(XS[[1:IP]])),
25 (XS[i] size(JQ) subarray of X(orig(IQ*(i-1)+1))
26     structure(blocks(H1,H2)),
27     H1 size(JQM) subarray of XS(orig(1))
28     structure(dense),
29     H2 size(JM) subarray of XS(orig(IQM+1))
30     structure(dense)
31 ) for i in [1:IP];
.
.
.
80 pcalc for j in [1:IP];
81 call DECOMP(IQM,IQM,A,AS[j],E1,WK,WKS[j],
      IPV,IPVS[j],EPSI);
82 call MULT(IQM,IQM,IQM,IM,
83     A,AS[j],E1,A,AS[j],G1,DMY,DMYS[j],
      IPV,IPVS[j]);
84 call MULT(IQM,IQM,IQM,IM,
85     A,AS[j],E1,A,AS[j],E2,DMY,DMYS[j],
      IPV,IPVS[j]);
86 call MULT(IQM,IQM,IQM,1,
87     A,AS[j],E1,X,XS[j],H1,DMY,DMYS[j],
      IPV,IPVS[j]);
88 end pcalc;
89 pcalc for j in [1:IP];
90 A,AS[j],G2=A,AS[j],G2-A,AS[j],E3*A,AS[j],G1;
91 A,AS[j],E4=A,AS[j],E4-A,AS[j],E3*A,AS[j],E2;
92 X,XS[j],H2=X,XS[j],H2-A,AS[j],E3*X,XS[j],H1;
93 end pcalc;
.
.
.
```

図-4 PARAGRAM による帯行列解法の記述

`pcalc for` はサブアレイ  $AS[j]$  に関してこれらを並列に処理することを指定している。`DECOMP` はサブアレイ内のブロック  $E1$  の LU 分解ルーチンであり、`MULT` は  $E1$  を単位行列化するのに必要な行列の積を計算するルーチンである。89 か

ら 93 行はフェーズ 1 のステップ 2 において各  $AS[j]$  の  $Ei3$  を並列に消去するための演算である。このように、行列の構造の定義とその構成要素であるサブアレイを使って並列アルゴリズムを明確に記述することができたと考える。

#### 4. 並列アーキテクチャと言語処理ソフトウェア

##### 4.1 並列アーキテクチャ

ここではこの例題を実行させたアーキテクチャについて述べる。このアーキテクチャは上記に述べた大型プロジェクトで開発されたものである。その構成は 4 台の PE (Processing Elements) が CMU (Common Mapping Unit) を介して大容量共有記憶装置 CSU (Common Storage Unit) に接続されている。各 PE はベクタおよびスカラ演算機構と固有の記憶装置 LS (Local Storage) をもっている。すなわち、2 レベルの階層記憶型の並列機である<sup>4), 5)</sup>。ソフトウェア側からみると、CSU に置かれたデータを各 PE から直接アクセスすることは一応可能ではあるが、性能上、実質的には各 PE の LS へデータを転送した後演算を施す必要がある。また、LS 上で更新したデータを他の PE でその後使用する可能性があれば CSU へ逆転送しておく必要がある。これらのデータ転送処理は当然言語処理ソフトウェアの責任となる。

##### 4.2 PARAGRAM 言語処理ソフトウェア

この言語で記述された応用コードから高い並列実行性能を得るには、その並列性をアーキテクチャへの確に写像する言語処理ソフトウェアが必要である。ここで述べる処理ソフトウェアはプロセッサ台数に特に依存するものではないが、対象応用プログラムの処理の粒度などから、一応数台～数十台程度のシステムを念頭に置いている。そのソフトウェア構成は図-5 のようである。各ソフトウェアモジュール間の共通インタフェースである中間語ファイルを作成する入力部、ハザード・デッドロックなどの並列処理特有の病理現象を検出する並列性静的検証部、プログラムの事前実行によって動的負荷情報を採取する動的解析部、動的負荷情報をもとにアーキテクチャへのマッピングを静的に行うタスク割付部、割付結果を Phil 言語 (本プロジェクトの基本並列処理記述言語<sup>5)</sup>) で表現する Phil プログラム生成部などからなる。

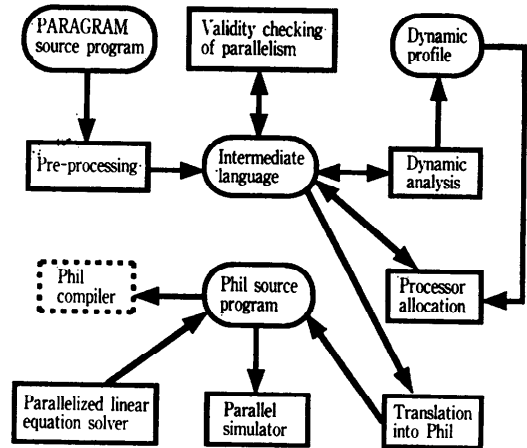


図-5 言語処理ソフトウェアの構成

##### 4.3 プロセッサ割付方式

これらのうち、並列処理性能を支配するのはタスク割付 (プロセッサ割付) である。これまでに、主に密結合マルチプロセッサに関して、いくつかのプロセッサ割付アルゴリズムが発表されている。対象並列構造についてみると、doall, doacross, serial loops のネストで構成されるものに対しては、イリノイ大学の Polychronopoulos による静的最適化割付アルゴリズム OPTAL、動的最適化割付アルゴリズム GSS など<sup>6)~8)</sup> がある。一方、parallel case タイプの、不均一サイズのタスク群で構成されるタスクグラフに対しては、笠原の静的割付アルゴリズム CP/MISF<sup>9)</sup> などがある。これに対して、PARAGRAM では多様な並列構造を許容するため、doall, doacross, serial loops と parallel case による任意の組合せによる多重並列構造の記述ができる。そのため、そのような組合せ構造に対応できる新たなプロセッサ割付方式が必要となり、最急降下法<sup>10)</sup>に基づく静的割付方式<sup>11)</sup>を考案した。

この方式は実行時のオーバーヘッドを抑止することを主眼として、静的割付を行っている。静的割付の効果を高める最適化のためには、動的負荷情報、すなわち、並列ループの繰返し回数および並列ループ本体の演算量などの情報を、実行に先だってあらかじめ得られることが望ましい。このため、本処理ソフトウェアでは上記に述べた動的解析機能でこれらの情報を採取して、これをプロセッサ割付機能へ入力する機構を有している。また、この基本アルゴリズムを実機のアーキテク

ャに対応できる実用的なものにするため、実機で採用されている階層記憶の利用で生ずるデータ転送オーバーヘッド、および各プロセッサエレメントのベクトル演算機構の加速性能の効果などを反映させる仕組みも設けた。また、本アルゴリズムでは、アレイデータは最初 CSU に配置しておき、タスクの起動（または終了）時に必要アレイを LS へ自動転送（または CSU へ自動逆転送）するようにしている。このため、データ転送オーバーヘッドを、本来の演算量の増加、およびそのタスクの開始時間の遅延として反映させる計算モデルを採用した。

## 5. 適用結果

### 5.1 PARAGRAM 言語による記述性

上記帯行列の並列解法プログラムを並列制御ライブラリ付きのある FORTRAN（いわゆるマクロタスキングレベルの機能をもつ）で書いた場合との比較を表-1 に示す。全体では記述行数は PARAGRAM の場合は約 1/2 で済んでいる。PARAGRAM ではデータの構造を明確にするため、この部分のデータ宣言の記述量は多いが、タスクの生成、起動、終了など並列制御の細かな指定は FORTRAN に比べ大幅に少なくなっている。また、実質的な演算の記述行数も、行列構造の宣言に合わせた配列演算を記述できることから、かなり削減されている。

### 5.2 実行性能解析

上記並列解法アルゴリズムの演算量と本プロジェクト機上でのデータ通信のコストを分析した結果を表-2 に示した。この表の値は、各フェーズの浮動小数点演算と、CSU-LS 間の浮動小数点

表-1 プログラム記述行数の内訳比較

|                                              | FORTRAN | PARAGRAM | PARAGRAM/<br>FORTRAN |
|----------------------------------------------|---------|----------|----------------------|
| Declaration for data structure               | 77      | 274      | 3.6                  |
| Specification related to parallelism control | 497     | 38       | 0.08                 |
| Intrinsic arithmetic operations              | 785     | 389      | 0.50                 |
| Total number of program lines                | 1,359   | 701      | 0.52                 |

Note: The FORTRAN shown above is a kind of parallel FORTRAN furnished with primitive parallel task control routines. (It can be considered a so-called macro task FORTRAN level.)

データの転送量、それと各フェーズのタスクの並列度の解析結果を、係数行列サイズ  $N=8192$ 、ブロック総数  $p=32$ 、 $2m=q=256$  に適用して得られた結果である。ここで、 $T_a$  は浮動小数点演算一回当たりの実行時間、 $T_c$  は浮動小数点データ 1 個当たりの転送時間である。 $T_c$  に関しては LS—CSU 間の双方向とも同一転送レートとする。また、データ転送は逐次的に行われ、 $T_c$  は演算時間に単純に加算して考えるものとした。他のフェーズに比べ、フェーズ 1—ステップ 3 の終わりでは係数行列の全ブロックを CSU へ戻す必要があるためデータ転送が多くなっている。なぜなら、フェーズ 2 では、巡回的に行列サイズを縮約する際、係数行列のブロックのプロセッサへの割付を各回ごとに変更する必要があるからである。並列に処理できるブロックの個数はフェーズ 2 での反復では次々と半減、フェーズ 3 での反復では次々と倍増するが、使用プロセッサ台数に対してブロック数が十分に大きければ高い並列度を保った計算ができる。

一方、表-3 はこの例題に対して PARAGRAM

表-2 並列実行時間の予測

| Phase in the algorithm                                                 | Serial execution time estimation | Parallel execution time estimation on 4 PEs |
|------------------------------------------------------------------------|----------------------------------|---------------------------------------------|
| Phase-1 (including step 1 2 3)                                         | $8.4E8 * T_a + 1.7E8 * T_c$      | $2.1E8 * T_a + 1.7E8 * T_c$                 |
| Phase-2 (including 4 times matrix reduction plus one LU decomposition) | $7.7E8 * T_a + 5.5E7 * T_c$      | $2.2E8 * T_a + 5.5E7 * T_c$                 |
| Phase-3 (including 4 times of Phase-3a plus once of Phase-3b)          | $4.0E6 * T_a + 2.0E6 * T_c$      | $1.1E6 * T_a + 2.1E6 * T_c$                 |
| Total                                                                  | $1.6E9 * T_a + 2.3E8 * T_c$      | $4.3E8 * T_a + 2.3E8 * T_c$                 |

Notes: (1)  $T_a$ : execution time per one floating-point operation

$T_c$ : communication time per one floating-point data (8 bytes)

(2) Matrix dimension:  $N=8192$ ,  $P=32$ ,  $q=N/P=256$ ,  $m=128$ ,  $s=q-m=128$

(3)  $T_c$  is assumed to be additive to  $T_a$  in both serial and parallel execution.

表-3 本プロジェクト機での帯行列解法実行性能

| Case-1 Matrix size | $N=8192$ | $P=32$ | $2m=q=256$ |
|--------------------|----------|--------|------------|
| Number of used PEs | 1        | 2      | 4          |
| Speedup ratio      | 1.00     | 1.81   | 2.83       |
| Case-2 Matrix size | $N=4096$ | $P=32$ | $2m=q=128$ |
| Number of used PEs | 1        | 2      | 4          |
| Speedup ratio      | 1.00     | 1.74   | 2.39       |
| Case-3 Matrix size | $N=4096$ | $P=16$ | $2m=q=256$ |
| Number of used PEs | 1        | 2      | 4          |
| Speedup ratio      | 1.00     | 1.53   | 2.09       |

- (1) The above measurement was performed on the Japanese National Research Project machine.  
 (2) The above values indicate relative speedup measured in elapsed time.

トランスレータが自動生成した並列化コードを実行して得た性能の相対比である。PE 台数 1, 2, 4 の場合を測定した。4 台の場合 (表-3 Case-1) は 1 台の場合の 2.83 倍の性能である。上記での予測において、もし  $T_a = T_c$  であるとした場合は 2.77倍となりこの実測とほぼ合致する値となる。

## 6. おわりに

以上の検討から、本言語 PARAGRAM による応用プログラムの記述性については、下記の特徴点がある程度確認できたと考える。

(1) 特徴機能の一つである階層的疎行列記述は、プログラミング経験上、並列構造の明確化に有効であり、適用範囲も広いと考える。

(2) FORTRAN などでの記述では必須である階層記憶間のデータ転送制御など、アーキテクチャに依存した記述は不要であり、計算アルゴリズムの見通しを良くしていると考えられる。

今回開発した言語処理ソフトウェアは、共有記憶並列を階層記憶並列へ自動変換するものであり、その方式を実証できた。しかし、ここでは、各 PE へのデータの分割配置に関しては必ずしも良い方式を採用しているとは言えない。それでも、本アーキテクチャ特性と大粒度並列を扱うという条件下では特に大きな問題とはならなかった。一方、今後各種の並列アーキテクチャが普及すると考えられる。特に、分散記憶型超並列機などに対しては、データ領域の最適分割・配置方式が本質的に性能を支配すると考えられる。この方面のソフトウェア技術の一層の研究開発が必要である。

**謝辞** 本研究は通商産業省工業技術院大型プロジェクトの一環として新エネルギー・産業技術総合開発機構 (NEDO) から委託を受けて実施したものである。本言語システムの試験に当たっては、(株)富士通 SSL 鈴木滋取締役、富士通(株)神谷幸男課長、橋本伸氏はじめ多くの方々にご協力いただいた。また、(株)日本科学技術研究所、(株)富士総合研究所の方々にはプログラム開発にご尽力いただいた。これらの方々には深謝いたします。

## 参 考 文 献

- 1) Yamamoto, F., Umetani, Y. and Demoto, M.: PARAGRAM: A High-Level Programming Language for Parallel Processors, Systems and Computers in Japan, Vol. 20, No. 8, pp. 100-109 (1989).
- 2) Jhonsson, S. L.: Solving Narrow Banded System on Ensemble Architectures, ACM Trans. Math. Soft., Vol. 11, No. 3, pp. 271-288 (Sep. 1985).
- 3) Yamamoto, F.: Solving Banded Systems using a Parallel Programming Languages with Hierarchically Data Descriptive Features, Proc. of the International Conference on Supercomputing, pp. 406-413 (1991).
- 4) 「科学技術用高速計算システム 研究開発」成果発表会講演予稿集, 科学技術用高速計算システム技術研究組合, pp. 171-184 (昭和 63 年 6 月).
- 5) 「科学技術用高速計算システムの研究開発」成果発表会論文集 II, 科学技術用高速計算システム技術研究組合, pp. 217-298 (平成 2 年 6 月).
- 6) Polychronopoulos, C. D., Kuck, D. J. and Padua, D. A.: Execution of Parallel Loops on Parallel Processor Systems, Proc. of ICPP '86, pp. 519-527 (1986).
- 7) Polychronopoulos, C. D. and Banerjee, U.: Processor Allocation for Horizontal and Vertical Parallelism and Related Speedup Bounds, IEEE Trans. on Computer, Vol. C-36, No. 4, pp. 410-421 (Apr. 1987).
- 8) Cytron, R.: Limited Processor Scheduling of Doacross Loops, Proc. of ICPP '87, pp. 226-234 (1987).
- 9) Kasahara, H. and Narita, S.: Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, IEEE Trans. Comput., C-33, pp. 1023-1029 (Nov. 1984).
- 10) Booth, A. D.: Numerical Methods, Butterworths (1959).
- 11) Yamamoto, F.: A Processor Allocation Algorithm for Nested Combination of Parallel Loops and Cases, Proc. of ICPP '89 (Vol. II), pp. 131-138.

(平成 3 年 10 月 23 日受付)



### 山本富士男 (正会員)

昭和 22 年生。昭和 45 年北海道大学理学部数学科卒業。同年(株)日立製作所に入社。以来、中央研究所にて、問題向け言語、回路シミュレーション、線形計算、並列記述言語、スケジューリングの研究に従事。現在同所主任研究員、情報処理学会誌編集委員。



### 出本 学 (正会員)

1981 年愛媛県立松山工業高校情報技術科卒業。同年(株)日立製作所入社。以来、同社中央研究所にて、日本語処理、並列処理ソフトウェア、ネットワーク利用環境などの研究開発に従事。



### 梅谷 征雄 (正会員)

1944 年生。1968 年東京大学理学部数学科卒業。同年(株)日立製作所に入社。以降、中央研究所にて、デジタル回路の故障診断、スーパーコンピュータ向き自動ベクトル化コンパイラ・偏微分方程式向き高水準言語 DEQSOL、並列プログラミング言語の研究に従事。現在の関心は、視覚的な数値シミュレーション環境と超並列計算機向きプログラミング手法にある。応用数理学会、ACM 各会員。

