データ再利用による分散メモリ上のデータ転送最適化手法

李 暁傑[†] 原田賢一[†]

分散メモリマシンはグローバルなアドレス空間を持たないため，アクセス対象となる配列要素が，自分自身のプロセッサ中のメモリに存在しないときは，データ転送が必要である．あるプロセッサにおいて，他のプロセッサのメモリ中にある配列要素への参照が起こったとき，その要素の値が以前の実行によって，すでに得られていることが保証できれば，データの再利用ができ，データ転送が避けられる．本論文は，データ転送の冗長性を完全冗長，条件付き冗長，および部分冗長の3種類に分類し，データフロー解析によって得られる結果をもとに，冗長なデータ転送を削除する手法を提案する．最初に，プログラムの実行にともなう配列要素へのアクセス状況を表すために，データの集合とそれらのデータをアクセスするプロセッサの集合との対応付けを示すデータアクセス記述子を導入する．この記述子を用いて，区間データフロー解析によって配列中で値が再利用できる部分を見つけ出す手法とデータフロー解析の結果に基づく最適化手法を示す．最後に，実験例によって最適化の効果を示す．

Removing Redundant Data Transmissions on Distributed Memories

XIAOJIE LI[†] and KEN'ICHI HARADA[†]

For distributed memory machines, the absence of global address space, and the need for explicit message passing among processors make such machines very difficult to write programs. In typical numerical applications, a significant portion of data transmissions arises from accesses to array sections. Data transmission overheads due to such traffic can be reduced by using compile-time information to detect data which can be reused. In this paper, we divide redundancy into completely redundant, conditionally redundant, and partially redundant, then present a data flow analysis framework for determining available array sections. A new kind of array section descriptor, called *Section Access Descriptor* is introduced, which represents the mapping of an array section onto the processor grid. Based on the interval data flow analysis, available array sections are determined, and transmissions corresponding these sections can be eliminated. Evaluation experiments show that this optimiztion technique is effective.

1. まえがき

分散メモリマシンでは，グローバルなアドレス空間が存在しないため，効率の良いプログラムを作成することが困難である．プロセッサ間でのデータ転送には，プロセッサ通信が必要となり，プログラムの実行効率は，そのコストによって左右される．そこで，最適な通信コードの生成を利用者から解放し，並列化コンパイラによって行うことが望まれる．並行プログラムの記述としては，データ分割の仕方を指示するためのディレクティブを，逐次プログラムの中に挿入した形式が主に用いられ，これまでに多数のコンパイラ[8),10),13)]が開発されている．それらのコンパイラでは，**所有者計算規則** (*owner computes rule*)[8),13)]とよばれる規則に従って，代入文の実行を担当する各プロセッサを決め，**SPMD** (*single-program multiple-data*) 実行方式の目的プログラムを生成している．ここで，所有者計算規則とは，代入文を実行する際，左辺のデータを**所有**しているプロセッサがその実行を行うことをいう．所有者計算規則に従って，プロセッサ P_i が代入文を実行する場合，右辺の評価に必要なデータが P_i のメモリになければ，その値を獲得するために通信によるデータ転送が必要となる．通信コードの最適化として，これまでにメッセージベクトル化 (*message vectorization*)[8)]，集団通信 (*collective communication*)[10)]，およびオーバラップ通信 (*overlapping communication*)[13)]などの手法が提案されている．しかし，これらの手法のほかに，次に述べるようにさらにデータ転送の最適化が考えられる．

代入文の右辺の評価において，プロセッサが所有していない配列要素への参照に対しては，従来，プロセッ

† 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University

サ通信を行うものとして扱ってきた．もし，それ以前の文の実行によって得られた値のコピーを保持しておけば，プロセッサ間でのデータ転送が不要になる場合がある．そのようなデータ転送をプログラムの静的解析によって検出するには，グローバルな範囲での配列要素の参照と代入の関係を解析する必要がある．

あるプロセッサでのプログラム実行において，所有していないデータがはじめて参照された場合，データ転送によってその値のコピーを得るものと仮定する．プロセッサ P_i 上の計算点 n において，データ x のコピーがすでに存在し，コピー後，n までの実行において x の値が更新されることがなければ，x は P_i の点 n で**コピー参照可能**という．あるプロセッサが所有しているデータの更新にともなって，コピー参照可能であったデータは**無効**になる．代入文の実行において，右辺に現れる配列要素 $A(I)$ を所有するプロセッサが，代入を行うプロセッサと異なっていても，$A(I)$ がコピー参照可能であれば，そのコピーを用いることによって，$A(I)$ の再送を避けることができる．

本論文では，High Performance Fortran (**HPF**)[6] で書かれたプログラムを対象として，データ転送の冗長性を完全冗長，条件付き冗長，および部分冗長に分類し，これらの冗長性を検出する手法を提案する．その検出には，仮想プロセッサ空間での検出と，実プロセッサ空間での検出の 2 通りの方式が考えられる．HPF では，配列の要素は，**整列** (`ALIGN`) と**配置** (`DISTRIBUTE`) によって 2 段階で実プロセッサに割り当てられるので，配列要素がどの実プロセッサ上にあるかを直接求めるには，複雑な処理が要求される．一方，配列要素がどの仮想プロセッサ上にあるかは，整列ディレクティブだけで決定されるので，配列要素の仮想プロセッサ上での位置をより簡単に求めることができる．この点に着目して，本論文では，仮想プロセッサ空間における冗長なデータ転送を明らかにし，その検出は，配列要素の実プロセッサへの配置を考慮すると，実装が容易になることを示す．

本論文の構成は次のとおりである．まず，2 章で，解析の対象にするプログラムと冗長性について述べ，この論文で用いるデータフロー方程式を示す．3 章では，プログラムの実行にともなってアクセスされる配列要素の集まり（以下，**配列セクション**または単に**セクション**という）と，それらの動作を行うプロセッサとの対応関係（以下，**セクションアクセス情報**という）を表すために**セクションアクセス記述子**とよぶ記述子を導入する．4 章と 5 章では，**区間データフロー解析法** (*interval analysis*)[11),14)] によるセクションアクセス情報を求める手順を与え，6 章で，冗長なデータ転送を検出する手法を示す．7 章では，本手法に対する評価の結果を示す．

2. 冗長性とデータフロー方程式

本研究では，HPF で書かれた整構造のプログラム単位を対象とし，手続き内データフロー解析 (*intra-procedural data flow analysis*) によってコピー参照可能な配列のセクションを求めることを考える．スカラ変数の配置に関しては，HPF で規定されていないので，スカラ変数に対する代入と参照は解析の対象から除く．これらの制約によって，本来冗長でなかったデータ転送を冗長なものとして検出してしまう可能性はない．

HPF において，配列は論理的な構造を持つデータの集まりとして扱われる．配列を分散メモリ上に割り付ける際，整列と配置のために，多次元の**仮想プロセッサ空間**が用意されている．本論文では，解析を容易にするために，1 つの仮想プロセッサ空間 (VPROCS) での最適化を考える．整列は，プログラムで使用する配列の各要素を VPROCS のどの要素に対応させるかを示し，配置は，`BLOCK(k)` と `CYCLIC(k)` とよばれる方式で，VPROCS の要素を実プロセッサと結び付ける．配列要素の添字や ALIGN パラメータには，ループ制御変数 (I) を用いて $\alpha \times I + \beta$ のような線形の式が多く用いられることに注目し，本論文では，添字の表現と ALIGN パラメータをこの形式に制限する．

図 1 に HPF のプログラム例を示す．このプログラムでは，HPF の整列化ディレクティブによって，配列要素 $A(I,J)$ と $B(I,J)$ とを同一の仮想プロセッサ VPROCS(I,J) に，すなわち A と B の各要素を仮想プロセッサ空間の同じ位置に整列させている．配列 C，D，E は VPROCS の第 1 列に整列させている．

2.1 データ転送の冗長性

本論文では，データ転送の冗長性を次の 3 種類に分類する．(1) ある計算点 n において，そこで参照される配列セクションのすべての要素が，コピー参照可能であれば，n におけるデータ転送は**完全冗長**であるという．(2) ある条件を満たすときに限って，コピー参照可能である場合，そのセクションに対する転送は**条件付き冗長**であるという．(3) 一部の配列要素がコピー参照可能である場合，n におけるそのセクションに対する転送は**部分冗長**であるという．

例 1. 図 1 のプログラムを用いて，冗長なデータ転送の例を示す．`s3` において，所有者計算規則から，$B(I,J)$ を所有する仮想プロセッサが，右辺を評価し代入を

```
      DIMENSION A(100,100),B(100,100)
      DIMENSION C(100),D(100),E(100)
      !HPF$ TEMPLATE VPROCS(100,100)
      !HPF$ ALIGN (I,J) WITH VPROCS(I,J)::A,B
      !HPF$ ALIGN (I) WITH VPROCS(I,1)::C,D,E

s1:     DO I = 2, 100
s2:       DO J = 1, 100
s3:         B(I, J) = ... A(J, I) ...
s4:       ENDDO
s5:       A(100, I-1) = ...
s6:       D(I) = A(1, I)
s7:     ENDDO
s8:     A(1, 2) = 2*B(2, 1)
s9:     IF (N .EQ. 100) THEN
s10:      DO I = 2, 100
s11:        C(I) = ... A(100, I) ...
s12:      ENDDO
s13:    ENDIF
s14:    DO I = 30, 50
s15:      E(I) = A(100, I)
s16:    ENDDO
s17:    DO I = 2, 100
s18:      E(I) = A(1, I) +10
s19:    ENDDO
```

図1 プログラム例
Fig. 1 A sample program.

行う．このとき，ALIGN ディレクティブの指定から，$I \neq J$ の場合には，$B(I,J)$ を所有するプロセッサと $A(J,I)$ を所有するプロセッサとは異なる．したがって，$B(I,J)$ への代入を行うプロセッサは，$A(J,I)$ の値をプロセッサ通信によって他のプロセッサから得る必要がある．s2 のループで参照された $A(J,I)$ のセクションの要素は，s6 の実行まで値が更新されることはない．D は B の第1列と同じ仮想プロセッサと対応付けられている．したがって，s6 の実行を考えると，s3 で $B(I,J)$ への代入を行った仮想プロセッサと，$D(I)$ への代入を行う仮想プロセッサとは，所有者計算規則から同一であり，s3 でデータ転送によって得られた要素 $A(1,I)$ はコピー参照可能であることが分かる．すなわち，s6 の実行において，$A(1,I)$ に対するデータ転送は完全冗長である．

s10 の DO 文は，s9 での条件 (N .EQ. 100) を満たすときに限って実行される．s11 の実行において，$A(100,I)$ の値は，$C(I)$ を所有するプロセッサ，すなわち VPROCS(I, 1) へ転送される．$E(I)$ は $C(I)$ と同じ仮想プロセッサと対応付けられているので，s15 の $E(I)$ への代入において，$A(100,I)$ はコピー参照可能である．したがって，s9 の条件が成立する場合，s15 における $A(100,I)$ に対する転送は条件付き冗長である．

また，s18 では，s17 による繰返し全体で $A(1,2:100)$ が参照される．その中で，$A(1,3:100)$ に対する転送は，s6 での転送によって部分冗長となる． □

3種類の冗長性の検出法と各冗長性に対するデータ転送の最適化手法については，6章で述べる．例1に示したように，プログラムの流れに沿って配列に対するアクセス状況を解析すれば，コピー参照可能なセクションを検出することができる．

2.2 データフロー方程式

解析対象となるプログラムのフローグラフを考え，その各節点でのセクションアクセス情報を表すデータフロー方程式を示す．本稿では，簡単のため，1つの配列に対するセクションだけを考える．複数の異なる配列が用いられている場合には，各配列を独立に扱い，それらに対するセクションアクセス情報を集合として扱えばよい．

フローグラフの開始節点から，ある節点 n への経路上で参照され，その後 n までに代入が行われない配列要素によって構成されるセクションを n における**参照セクション**とよぶ．また，開始節点から n への経路上のある節点で代入が行われ，その後 n までに参照されることがない配列要素によって構成されるセクションを，n における**無効セクション**とよぶ．ある条件が成り立つ場合に限って参照されるセクション，および無効にされるセクションを，それぞれ**条件付き参照セクション**，**条件付き無効セクション**とよぶ．

フローグラフの各節点における参照セクションと無効セクションの一般的な関係を以下に述べる．まず，各節点 n_i について，次の集合を定義する．

- IN_i (IN_i^{con})：n_i の入口における（条件付き）参照セクションの集合
- OUT_i (OUT_i^{con})：n_i の出口における（条件付き）参照セクションの集合
- K_i (K_i^{con})：n_i の出口における（条件付き）無効セクションの集合
- Gen_i (Gen_i^{con})：n_i の実行によって，(条件付き) 参照されるセクションの集合
- $Kill_i$ ($Kill_i^{con}$)：n_i の実行によって，(条件付き) 無効になるセクションの集合
- $UPIN_i$：n_j を n_i の後続節点とすると，n_j で参照されるセクションの中で，そのデータ転送を n_i

の入口に移すことが可能なセクションの集合

- $UPOUT_i$：n_j を n_i の後続節点とすると，n_j で参照されるセクションの中で，そのデータ転送を n_i の出口に移すことが可能なセクションの集合

節点 n_i についてのデータフロー方程式は，次に示すとおり，従来のデータフロー問題における方程式と同じ形の式で表すことができる[1]．方程式中の p は n_i の直接先行節（*immediate predecessor*）を表す．

完全冗長を求めるための方程式

$$IN_i = \cap_p OUT_p \quad (1)$$

$$K_i = (\cup_p K_p) \cup Kill_i \quad (2)$$

$$OUT_i = (IN_i - Kill_i) \cup Gen_i \quad (3)$$

条件付き冗長を求めるための方程式

$$IN_i^{con} = \cup_p OUT_p^{con} \quad (4)$$

$$K_i^{con} = (\cup_p K_p^{con}) \cup Kill_i^{con} \quad (5)$$

$$OUT_i^{con} = (IN_i^{con} - Kill_i^{con}) \cup Gen_i^{con} \quad (6)$$

続いて，$UPIN_i$ および $UPOUT_i$ の導入理由を明らかにするために，部分冗長なデータ転送の削除について述べる（以下，部分冗長なデータ転送の削除を**部分冗長性の解消**という）．ある節点 n_i の入口における参照セクション IN_i の一部が $Kill_i$ によって無効にされ，後続節点 n_j でその部分を参照しているときに，n_j において部分冗長性が生じる．その場合には，無効にされる部分の転送を n_i の出口で行うことによって，後続節点における部分冗長性を解消することができる．たとえば，例1では，s8 の出口で $A(1,2)$ を転送することによって，s18 での部分冗長性を解消できる．

部分冗長性がある場合，その節点の入口で，必要な要素についてだけデータ転送を行っても，冗長性を解消することができる．しかし，ここでは，転送コードをできるだけ上方の先行節点に移すことを考える．その理由は，1つの経路上だけではなく，他の経路上での部分冗長性の解消に役立つ可能性があるからである．次のプログラム片を考える．

```
s20:        A(50) = 100
s21:        IF (N .EQ. 50) THEN
s22:          DO I = 1, 90
s23:            C(I) =  A(I) + 10
s24:          ENDDO
s25:        ENDIF
s26:        DO I = 1, 80
s27:          E(I) = A(I) * 3
s28:        ENDDO
```

s26 のループを通して，s27 で参照されるセクションは $A(1:80)$ である．条件 (N .EQ. 50) が成立する場合，s22 のループを通して，s23 で参照されるセクションは $A(1:90)$ である．s20 の入口における参照セクションが $IN_{20} = A(1:100)$ であったとすると，s20 での代入によって，その中の $A(50)$ が無効にされる．このとき，s27 の入口で $A(50)$ の転送を行ってもよいが，$A(50)$ の転送を s20 の出口に移すことによって，s27 での部分冗長と，もう1つの経路（s23）での部分冗長が1回のデータ転送によって解消できる．

部分冗長性の解消にあたっては，どのセクションに対応するデータ転送を先行節点に移動できるかを求める必要がある．節点 n_i における集合 $UPIN_i$ と $UPOUT_i$ は，n_i の後続節点 n_j でのデータ転送を，それぞれ n_i の入口，出口に移動しても，n_j の実行に影響を与えない参照セクションを表す．

部分冗長を求めるための方程式

$$UPIN_i = (UPOUT_i - Kill_i) \cup Gen_i \quad (7)$$

$$UPOUT_i = \cap_s UPIN_s \quad (8)$$

ここで，s は n_i の直接後続節（*immediate successor*）を表す．

上述の関係に基づいて，コピー参照可能なセクションを求めるためには，ループ中の代入文について，代入あるいは参照されるセクションとその動作を行うプロセッサとの対応関係を維持しながら，解析をすすめる必要がある．そのために，次章でセクションアクセス記述子とよぶ表現法を導入し，上に示した集合をこの記述子で表すことを考える．

3. セクションアクセス記述子

SPMD によるプログラム実行の各計算点において，配列のどのセクションが参照され，あるいはどのセクションに代入が行われるか，すなわちセクションと，そのセクションをアクセスするプロセッサとを結び付けて表現する方法を考える．そのために，セクションアクセス記述子（*section access descriptor*，以下，略して **SAD**）とよぶ表現法を導入する．データフロー解析においては，各節点について，この記述子によって表される Gen_i と $Kill_i$ を求めておき，これらの値をもとに上記のデータフロー方程式を解く．この記述子によって表される IN_i と OUT_i が最終的に求めたい値である．なお，データフロー方程式に SAD に対する演算について，文献18)を参照されたい．

ある配列セクションを表す SAD は，その要素またはそれらの集まりを表す記述子（以下，**セクション記述子**という）D と，D によって示されるセクションをアクセスする仮想プロセッサを与える記述子（マッ

ピング記述子）M からなる．それを $\langle D, M\rangle$ で表す．D に M を適用すること，すなわち $M(D)$ によって，D をアクセスするプロセッサが得られる．

3.1 セクション記述子

n 次元配列 A に関するセクション記述子 D は次の形式で表す．

$A(e_1, e_2, \ldots, e_n)$

ここで，A は配列名であり，$e_1, e_2, \ldots$ はそのセクションを表す次の形の式とする．

(1) $\boldsymbol{\alpha \times k + \beta}$：$k$ はループ制御変数，α，β は定数とする．

(2) $\boldsymbol{l : h : s}$：l から h までの s 刻みによる整数の集合を表す．各パラメータは定数とする．以下，この形の式を範囲表現とよぶ．$s = 1$ のときは，$l : h$ で表す．

(3) $\boldsymbol{\phi}$：添字が，上の形以外の式のであることを表す．

ループ中に現れる配列要素に関して，ある繰返しでのアクセスを考えるときには，(1) の形式を用い，ループの繰返し全体を通してアクセスされるセクションを表すときには，ループ制御変数を繰返しパラメータで置き換えた (2) の形式を用いる．

3.2 マッピング記述子

マッピング記述子 M は，2 つのベクトル P，F の対 $\langle P, F\rangle$ で表す．それぞれを**次元整列ベクトル**，**マッピング関数ベクトル**とよび，それらは仮想プロセッサ空間の次元数と等しい長さ m を持つものとする．n 次元配列 A に対する各ベクトルの構成は次のとおりである．

$P = [P_1, P_2, \ldots, P_m]$ とすると，P の各要素 $P_i = k$ $(1 \le i \le m,\ 1 \le k \le n)$ は，仮想プロセッサ空間の第 i 次元が配列 A の第 k 次元と対応することを表す．配列の次元数が仮想プロセッサ空間の次元数より少ない場合には，仮想プロセッサのある次元 (i) が，配列のどの次元とも対応しないことがある．それを $P_i = \phi$ で表す．配列要素の仮想プロセッサ空間における各次元上での位置は F によって与える．

$F = [F_1, F_2, \ldots, F_m]$ とし，P_i の値を k とすると，要素 $F_i(j)$ $(1 \le i \le m)$ は，D によって示されるセクションに関して，配列 A の第 k 次元上の j 番目の要素，すなわち位置 j の要素が，仮想プロセッサ空間の第 i 次元上のどの位置に対応するかを示す関数である（例 2 参照）．この関数を，以降，**マッピング関数**とよぶ．マッピング関数 $F_i(j)$ は，次のいずれかの形式で表される．

$$F_i(j) = \begin{cases} c \times j + l : c \times j + u : s & \text{(a)} \\ \phi & \text{(b)} \end{cases}$$

上の式 (a) において，コロンで区切られた 3 つの項は，左から順にそれぞれ初期値，終値，増分を表す．(a) はそれらのパラメータによって規定される位置（整数）の集合を意味する．ただし，c，l，u および s はそれぞれ定数とする．$l = u$ のとき，一対一のマッピングを意味し，単に $F_i(j) = c \times j + l$ で示す．$u \ge l + s$ のとき，1 つの配列要素が複数の仮想プロセッサにマッピングされる．ステップ s が 1 のときは，式 (a) 中の s の表記を省略する．式 (b) は，マッピング関数が線形関数として定義できないことを表す．

$F_i(j)$ の値が非整数になる場合，配列 A の第 P_i 次元上の j 番目の要素が仮想プロセッサにマッピングされないことを意味する．SAD によって示されるセクションが無効になった場合，そこに含まれる要素は，すべてのプロセッサでコピー参照ができなくなる．その場合には，マッピング記述子 M を T によって示す．SAD の例を次に示す．

例 2. 2 次元の仮想プロセッサ空間 VPROCS を考え，配列 A のセクションへの参照が次の SAD によって示されたとする．

$\langle A(1:10,\ 1:20),\ \langle [2, 1],\ [F_1(j), F_2(j)]\rangle\rangle$

ここで，$F_1(j) = 1 : 10$，$F_2(j) = 2 \times j$ とする．

$P = [2, 1]$ は，配列 A の第 2 次元を仮想プロセッサの第 1 次元に対応させ，配列の第 1 次元を仮想プロセッサの第 2 次元に対応付けることを表す．このとき，$A(i, j)$ $(1 \le i \le 10,\ 1 \le j \le 20)$ は仮想プロセッサ VPROCS$(1 : 10,\ 2 \times i)$ によって参照されること，すなわちセクション $A(i,\ 1 : 20)$ が，10 個の仮想プロセッサ VPROCS$(1 : 10,\ 2 \times i)$ によって参照されることを表す．この関係を**図 2** に示す．この図では，$A(i,\ 1 : 20)$ を A_i で示す．図 2 (a) の各網をかけた部分がセクションを示し，図 2 (b) は仮想プロセッサ空間におけるプロセッサと A の各セクションの参照関係を示す．□

4. Gen_i と $Kill_i$ の SAD 表現

2 章に示したデータフロー方程式において，未知数は IN_i と OUT_i であり，Gen_i と $Kill_i$ は定数である．この章では，1 つの代入文がフローグラフの 1 つの節点 n_i を構成するものとして，n_i に対応する文から，SAD 表現による Gen_i と $Kill_i$ を得るための方法を述べる．Gen_i^{con} と $Kill_i^{con}$ も同じ方法で得られる．

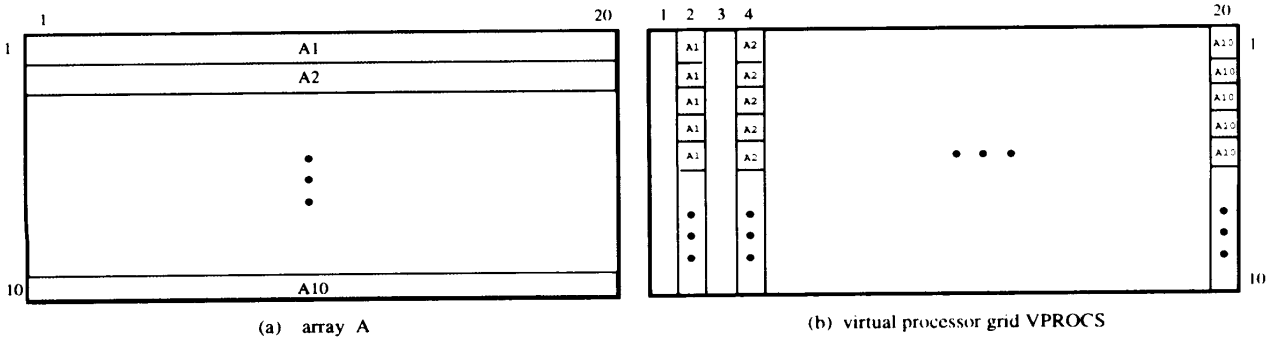

図2 SADによる配列要素のマッピング
Fig. 2 An example for SAD.

4.1 Gen_i と $Kill_i$

$Kill_i$ は節点 n_i における配列要素の無効状況を表すSADである．$Kill_i = \langle D^L, M^L \rangle$ とすると，セクション記述子 D^L は，左辺の配列要素をそのまま用いる．代入によって，その要素はすべてのプロセッサで無効になるため，M^L は $\top$ とする．すなわち，$Kill_i = \langle D^L, \top \rangle$ とする．

一方，$Gen_i = \langle D^R, M^R \rangle$ は，節点 n_i で右辺の配列要素のコピーの有効性を表すSADである．代入文の右辺において k 個の配列要素が参照される場合，それぞれに対応する参照セクションを $Gen_1, Gen_2, \ldots, Gen_k$ とすると，この節点における Gen_i は $\cup_{h=1}^{k} Gen_h$ で表される．以下では，簡単のため，1つの配列要素に対する Gen_i を考える．

セクション記述子 D^R には，右辺で参照される配列要素をそのまま用いる．マッピング記述子 $M^R = \langle P^R, F^R \rangle$ は，D^R を参照する仮想プロセッサ，すなわち D^R のコピーを保持する仮想プロセッサを，D^R から求めるための記述子である．D^R に対する次元整列ベクトル P^R とマッピング関数ベクトル F^R の意味とその求め方は次のとおりである．

以下の説明では，左辺の配列要素を所有するプロセッサを表すためのSADを $\langle D^L, Owner\_M^L \rangle$ で表し，$Owner\_M^L = \langle Owner\_P^L, Owner\_F^L \rangle$ で表す．このとき，$Owner\_P^L$ と $Owner\_F^L$ は ALIGN ディレクティブから直接求めることができる．まず，次の例で，F^R の意味を述べる．

```
!HPF$ ALIGN B(J) WITH VPROCS(c*J+1)
......
B(α1*K+β1) = A(α2 *K+β2) + 1
```

左辺の配列要素の添字の値を J とすると，$Owner\_F^L(J) = c \times J + l$, $Owner\_P^L = [1]$ となる．F^R は，上の代入文の右辺の配列要素 $A(\alpha_2 * K + \beta_2)$ のコピーがどの仮想プロセッサ上にあるかを示すマッピング関数である．$I = \alpha_2 * K + \beta_2$ とすると，上の代入文は次のように書き換えることができる．

$$B(\frac{\alpha_1 \times (I - \beta_2)}{\alpha_2} + \beta_1) = A(I) + 1$$

$A(I)$ の値は左辺の配列要素を所有するプロセッサへ転送されるので，$Owner\_F^L(J)$ によって示される対応関係から，右辺の配列要素 $A(I)$ のコピーは，仮想プロセッサ $F^R(I)$ 上に保持される．

$$F^R(I) = c \times (\frac{\alpha_1 \times (I - \beta_2)}{\alpha_2} + \beta_1) + l$$

ループ制御変数を K として，多次元配列の要素が代入文の左辺と右辺に現れる場合，K が用いられる添字の位置は必ずしも同じではない．その場合には，K を添字に含む右辺の配列要素の次元と，K を添字に含む左辺の配列要素の次元とを仮想プロセッサ空間の同一の次元に対応させてから，マッピング関数を適用する必要がある．

右辺の配列要素に対する次元整列ベクトル P^R はこのような対応付けを表す．次の例を考える．

```
!HPF$ ALIGN B(I,J,K,L) WITH VPROCS(I,J,K,L)
......
B(I, J, K, L) = A(I, K, J, 20) + 1
```

左辺の配列の次元と仮想プロセッサ次元との対応は，ALIGN ディレクティブから，$Owner\_P^L = [1,2,3,4]$ となる．$Owner\_P^L$ から，右辺の配列の第1次元 (I)，第2次元 (K)，第3次元 (J) は，それぞれ仮想プロセッサ空間の第1次元，第3次元，第2次元に対応する．右辺の配列の第4次元 (20) は仮想プロセッサの残りの第4次元に対応させる．したがって，$P^R = [1,3,2,4]$ となる．上述の考えをもとに，P^R と F^R の求め方を次に示す．

4.2 P^R の求め方

次の手順によって，右辺の配列の次元と仮想プロセッサ空間の次元との対応付けを行う．

(1) 仮想プロセッサ空間の次元数を m とするとき，長さ m のベクトル P^R を用意し，最初に各要素を ϕ にする．

(2) ループ制御変数 k が，左辺の配列要素の l 番

表1 右辺の配列要素に対するマッピング関数の構成規則
Table 1 The rule of calculating mapping functions for Gen_i.

$Owner\_P_i^L$-th Subscript	P_i^R-th Subscript	$F_i^R(j)$
$\alpha_1 \times k + \beta_1$	$\alpha_2 \times k + \beta_2$, $\alpha_2 \neq 0$	$c \times (\frac{\alpha_1 \times (j-\beta_2)}{\alpha_2} + \beta_1) + l$
	β_2	$c \times (\alpha_1 \times k + \beta_1) + l$
	ϕ	
ϕ	$\alpha_2 \times k + \beta_2$	l

目の添字として，$\alpha_1 \times k + \beta_1$（$\alpha_1$ と β_1 は定数）という形で用いられ，同様に，右辺でも h 番目の添字として，$\alpha_2 \times k + \beta_2$（$\alpha_2$ と β_2 は定数）という形で用いられているとする．左辺の配列の第 l 次元が仮想プロセッサ空間の第 i 次元に対応，すなわち $Owner\_P_i^L = l$ のとき，同じく右辺の配列の第 h 次元を仮想プロセッサ第 i 次元に対応付ける，すなわち，$P_i^R = h$ とする．

(3) 右辺の配列要素の添字を先頭から調べていって，h 次元目が仮想プロセッサ空間の次元と対応付けられていないとき，P^R 中の最初の空の要素 P_i^R に h を入れる．

4.3 F^R の求め方

右辺の配列要素の値を左辺の配列要素と同じ仮想プロセッサへマッピングするための関数を，次に述べる手順によって決定する．F^R は，仮想プロセッサ空間の次元数と同じ長さを持つマッピング関数のベクトルとする．

(1) 仮想プロセッサ空間の次元を $i\ (=1,2,\ldots,m)$ とし，$Owner\_M^L$ のマッピング関数を $Owner\_F_i^L(j) = c \times j + l$ とする．このとき，D^L 中 $Owner\_P_i^L$ 番目の添字，同様に D^R 中の P_i^R 番目の添字，および $Owner\_F_i^L(j)$ から，**表1** に示す関数 $F_i^R(j)$ が得られる．

(2) 上の条件を満たさない場合は，マッピング関数が構成できないことを示すために $F_i^R = \phi$ とする．

表1の2行目では，$F_i^R(j)$ が $c \times (\alpha_1 \times k + \beta_1) + l$ の形で表されている．これは，$F_i^R(j)$ の値が関数のパラメータ j に依存せず，ループ変数に依存することを意味する．ループ変数 k が K_1 から K_2 まで変化する場合，$F_i^R(j)$ は $c \times (\alpha_1 \times K_1 + \beta_1) + l : c \times (\alpha_1 \times K_2 + \beta_1) + l$ となる．

例 3. 次のプログラム片を考え，ループ中の代入文における $Kill_i$ と Gen_i を示す．

```
!HPF$ ALIGN (I,J) WITH VPROCS(I,J) :: A
......
A(I, J) = C(2*I, J-1) + 1
```

代入によって $A(I, J)$ が無効となるので，$Kill_i$ は $\langle A(I,J), \top \rangle$ となる．次に，Gen_i を求める．左辺の配列要素 $A(I, J)$ の位置については，**ALIGN** ディレクティブから，$Owner\_M^L = \langle [1,2], Owner\_F^L \rangle$ が得られる．ここで，$Owner\_F^L$ のマッピング関数は，それぞれ $Owner\_F_1^L(j) = j$，$Owner\_F_2^L(j) = j$ となる．

右辺の配列要素 $C(2*I, J-1)$ のマッピング記述子を $M^R = \langle P^R, F^R \rangle$ とする．まず，上述の方法によって，$P^R = [1,2]$ が得られる．次に，F^R の求め方に従って，マッピング関数を求める．$Owner\_P_1^L = 1$ であるから，仮想プロセッサ空間の第1次元が添字 I に対応し，$P_1^R = 1$，および $Owner\_F_1^L(j) = j$ から，表1の1番目の規則によって，$F_1^R(j) = 1 \times (\frac{1 \times (j-0)}{2} + 0) + 0 = \frac{j}{2}$ が得られる．プロセッサ空間の第2次元については，同じ方法で $F_2^R(j) = j + 1$ が得られる．したがって，Gen_i は $\langle C(2*I, J-1), \langle [1,2], [F_1^R(j), F_2^R(j)] \rangle \rangle$ となる． □

5. 区間データフロー解析法による参照セクションの求め方

この章は，区間分割に基づくデータフロー解析によって，フローグラフの各節点 n_i における参照セクションの集合を求めるための方法を示す．

5.1 区間データフロー解析法の概要

区間分割によって得られる1つの**区間**（*interval*）は，節点の集合からなり，各区間には**ヘッダ節点**（*header node*）とよばれる節点が必ず1つ存在する．本論文では，文献14)に従って，ループだけを区間として扱う．その場合，ループの最後の文にあたる節点を**最終節点**（*last node*）とよび，最終節点からヘッダ節点への辺を**逆順辺**（*inverse edge*）とよぶ．

区間分割においては，与えられたフローグラフ G をいくつかの区間に分割したあと，各区間を**要約節点**（*summary node*）とよぶ節点で代表させた高次のグラフ G^1 を作成する．そのグラフを**区間グラフ**（*interval graph*）という．さらに，G^1 を区間分割することによって，G^2 が得られる．対象としているプログラムが整構造を持つ場合，そのフローグラフは可約となる．可

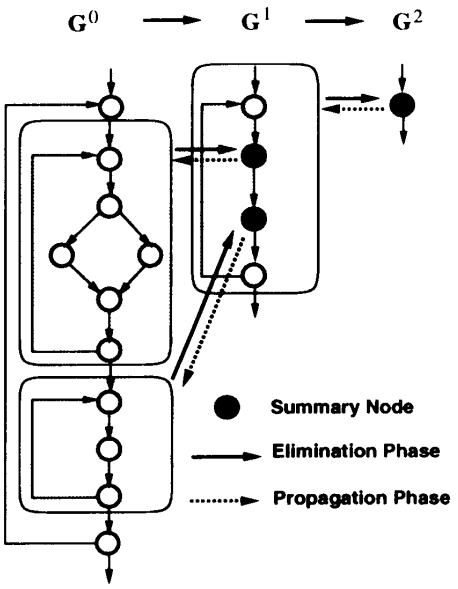

図3 区間データフロー解析
Fig. 3 Interval based data flow analysis.

約グラフの場合は，区間グラフにループが含まれなくなるまで，区間分割を繰り返すことによって，有限長の区間グラフ列 $G^0(=G), G^1, \ldots, G^n$ が得られる．

区間分割に基づくデータフロー解析は，次に示す簡約フェーズ（*elimination phase*）と伝播フェーズ（*propagation phase*）の2つのフェーズからなる．各フェーズにおける処理の概要は次のとおりである（**図3**参照）．

(1) 簡約フェーズ：区間グラフ列を $G^0\ (=G), G^1, \ldots, G^n$ とすると，この順に，区間グラフを処理する．そのとき，$G^i\ (i=0,1,\ldots,n-1)$ 内の各区間ごとに，その中で発生するローカルなデータフロー情報を収集し，その結果を G^{i+1} における要約節点に付与する．この処理によって，逆順辺を経てループの入口に到達する情報が要約節点に得られる．

(2) 伝播フェーズ：まず，G^n での各節点における IN_i と OUT_i を求めておく．次に，簡約フェーズとは逆に，すなわち，$G^{n-1}, G^{n-2}, \ldots, G^0$ の順に区間グラフを処理していく．区間グラフ G^i におけるある区間 I を考えたとき，I の外部から I のヘッダ節点に到達する情報は，G^{i+1} における I の要約節点に収集されている．そこで，その情報を区間内の各節点に伝えていくことによって，各節点におけるグローバルな情報が得られる．最終的に，G の各節点の入口，出口に達する情報が得られる．

5.2 簡約フェーズ

区間グラフ列を $G^0, G^1, \ldots, G^n$ とし，G^0 は，解析対象として与えられたプログラムに対するフローグラフ G とする．このフェーズでの処理を開始するとき，G の各節点 n_i では，4章で述べた方法によって，すでに Gen_i と $Kill_i$ が求められているものとする．簡約フェーズにおける処理は次のとおりである．

for $u := 0$ **to** n **do**
区間グラフ G^u が m 個の区間 $I_1^u, I_2^u, \ldots, I_m^u$ に分割されているものとする．
for $v := 1$ **to** m **do**
begin
1. 区間 I_v^u に含まれる節点を前向きに（制御の流れに沿って）たどりながら，各節点 n_i におけるローカルな無効セクション K_i' と参照セクション OUT_i' を，次のようにして求める．
1.1 I_v^u のヘッダ節点 n_h の出口では，参照セクションも無効セクションもないものとして，初期設定 $K_h' = \phi$，$OUT_h' = \phi$ を行う．
1.2 I_v^u 内の各節点 n_i を前向きにたどって，データフロー方程式(1)～(3)に従って，n_i での Gen_i と $Kill_i$ とから，K_i'，IN_i'，および OUT_i' を表す式を生成する．
2. 最終節点 n_l における K_l' と OUT_l' から，G^{v+1} における I_v^u の要約節点 s での Gen_s，$Kill_s$ を設定する．
end
end

ステップ2において，OUT_l' と K_l' を，I_v^u の要約節点 n_s の Gen_s，$Kill_s$ にそれぞれ与える際には，I_v^u に対するループ制御変数を k とすると，セクション記述子の中で，k によって表される添字の部分を k の制御パラメータでの表現（範囲表現）に置き換える必要がある．この操作は，k によって制御されるループの実行全体を通じての参照および無効セクションを求めることを意味する．そのために，次の関数 $iterations(S, k, low : high : step)$ を定義する．

関数 $iterations(S, k, low : high : step)$ は，与えられたSADの集合Sに対して，S中の各セクション記述子の添字 $\alpha \times k + \beta$ を次のように書き換えたSADを返す関数とする．

$$\alpha \times low + \beta : \alpha \times high + \beta : \alpha \times step$$

この書換えにともなって，仮想プロセッサ空間での位置を与えるマッピング関数を次のように置き換える．上に示した式が S のセクション記述子の中で t

表 2 図 1 の配列 A に対する簡約結果
Table 2 Elimination phase for array A in Fig. 1.

Step	SAD set	D	P	F
Elimination Step 1	OUT'_3	$A(J,I)$	2,1	$F_1(j)=j,\ F_2(j)=j$
	K'_3	ϕ		
	Gen_{s2}	$A(1:100,I)$	2,1	$F_1(j)=j,\ F_2(j)=j$
	$Kill_{s2}$	ϕ		
Elimination Step 2	OUT'_5	$A(1:100,I)$	2,1	$F_1(j)=j,\ F_2(j)=j$
	K'_5	$A(100,I-1)$	$\top$	$\top$
	OUT'_6	$A(1:100,I)$	2,1	$F_1(j)=j,\ F_2(j)=j$
	K'_6	$A(100,I-1)$	$\top$	$\top$
	Gen_{s1}	$A(1:99,2:100), A(1:100,100)$	2,1	$F_1(j)=j,\ F_2(j)=j$
	$Kill_{s1}$	$A(100,1:99)$	$\top$	$\top$
Elimination Step 3	$OUT^{con'}_{11}$	$A(100,I)$	2,1	$F_1(j)=j,\ F_2(j)=j$
	$K^{con'}_{11}$	ϕ		
	Gen^{con}_{s3}	$A(100,2:100)$	2,1	$F_1(j)=j,\ F_2(j)=j$
	$Kill^{con}_{s3}$	ϕ		
Elimination Step 4	OUT'_{15}	$A(100,I)$	2,1	$F_1(j)=j,\ F_2(j)=j$
	K'_{15}	ϕ		
	Gen_{s4}	$A(100,30:50)$	2,1	$F_1(j)=j,\ F_2(j)=j$
	$Kill_{s4}$	ϕ		

番目の添字として用いられたとする．その記述子に対応する次元整列ベクトルを P とすると，$P_i=t$ のとき，マッピング関数 $F_i(j)=c\times j+l$ は，t 番目の添字の値から，仮想プロセッサの i 次元上の位置を与える．t 番目の添字は，$\alpha\times low+\beta\ (=low')$ から $\alpha\times high+\beta\ (=high')$ まで，$\alpha\times step\ (=step')$ ずつ変わるので，$F_i(j)$ もそれに対応する値が得られるように，次の範囲表現で置き換える．

$$F_i(j)=c\times low'+l:c\times high'+l:c\times step'$$

区間 I^u_v の要約節点 n_s における Gen_s と $Kill_s$ は次の式で与えられる．ただし，そのループ制御変数を k とし，k の制御パラメータを $low:high:step$ とする．

$$Gen_s=iterations(OUT'_l,k,low:high:step)-(\cup_{def}iterations(S_{def},k,low:high:step)) \quad (9)$$

$$Kill_s=iterations(K'_l,k,low:high:step) \quad (10)$$

ここで，def は区間内の逆依存を表し，S_{def} は逆依存の定義側における配列要素の SAD を表す．逆依存が存在する場合，それまでの繰返しで参照していた要素は，次の繰返しで値の更新が行われるため，その要素は無効になる．したがって，Gen_s の計算においては，逆依存によって無効になる要素を除かなければならない．

例 4. 図 1 のプログラムで用いられている配列 A についての簡約結果を**表 2** に示す．ここでは，フローグラフの節点の表示にもとのプログラムの行番号を用いる．また，文 **s1**，**s2**，**s10**，および **s14** によって構成されるループをそれぞれ L_1，L_2，L_3，および L_4 とし，それらの区間に対する要約節点をそれぞれ n_{s1}，n_{s2}，n_{s3}，および n_{s4} で表す．

まず，L_2 に対する Gen_{s2} では，その参照セクションとして $A(1:100,\ I)$ が得られる．

ループ L_1 では，節点 **s5** で $A(100,\ I-1)$ の値が更新される．この代入によって，Gen_{s2} が無効になることはないので，節点 **s6** の出口における参照セクションは，$A(1:100,\ I)$ と $A(1,\ I)$，すなわち $A(1:100,\ I)$ となる．この結果から，Gen_{s1} は次のようにして求められる．

(1) $iterations(OUT'_6,I,2:100)$，すなわち OUT'_6 に対して，ループ制御変数 I を範囲表現に変換し，$A(1:100,\ 2:100)$ が得られる．

(2) **s5** によって，$A(100,I-1)$ が無効になる．

(3) $A(1:100,\ 2:100)$ への参照と，文 **s5** での $A(100,I-1)$ への代入との間には，逆依存の関係がある．$A(100,I-1)$ に拡張関数を適用して得られた部分，すなわち $A(100,1:99)$ を $A(1:100,\ 2:100)$ から除くことによって，表 2 に示す Gen_{s1} の結果が得られる．

L_3，L_4 に対する Gen^{con}_{s3} と Gen_{s4} は，それぞれ $A(100,2:100)$，と $A(100,30:50)$ となる． □

5.3 伝播フェーズ

各要約節点に集めた参照セクションの情報を，対応する区間のヘッダ節点に与え，その情報を区間内の各節点に伝播させる．この操作を，G^n から G^0 へ向かって繰り返すことによって，最終的に G の各節点 n_i で

表 3 図 1 の配列 A に対する伝播の結果
Table 3 Propagation phase for array A in Fig. 1.

SAD set	Equation	D
$IN_2(I)$	$OUT_1(I) = IN_1(I)$	$A(1:99, 2:I-1), A(1:100, I-1)$
$OUT_2(I)$	$(IN_2 - Kill_{s2}) \cup Gen_{s2}$	$A(1:99, 2:I-1), A(1:100, I-1:I)$
$IN_5(I)$	$OUT_2(I)$	$A(1:99, 2:I-1), A(1:100, I-1:I)$
$OUT_5(I)$	$(IN_5 - Kill_5) \cup Gen_5$	$A(1:99, 2:I), A(1:100, I)$
$IN_6(I)$	$OUT_5(I)$	$A(1:99, 2:I), A(1:100, I)$
$IN_{11}^{con}(I)$	$OUT_{10}^{con}(I) = IN_{10}^{con}(I)$	$A(100, 2:I-1)$
$IN_{15}(I)$	$OUT_{14}(I) = IN_{14}(I)$	$A(1:99, 2:100), A(1:100, 100)$ $A(100, 30:I-1)$

の参照セクションの集合 IN_i, OUT_i が得られる.

区間グラフ G^u におけるある区間 I_v^u を考え, そのループの制御変数を k とする. このとき, I_v^u 内の各節点 n_i での IN_i, OUT_i は, 一般に k のある繰返しまでに参照されるセクションを表すようにする. そのために, まず, ヘッダ節点 n_h における IN_h を関数 $IN_v^u(j)$ として定義する. $IN_v^u(j)$ は, k の制御パラメータを low, $high$, $step$ とし, $k = low, \ldots, j-step$ の実行を終え, $k = j$ となったときの, 区間 I_v^u のヘッダ節点 n_h における参照セクションを表す.

$IN_v^u(j)$ は, (1) ループに入る前で参照され, 繰返し $k = low : j - step$ で無効にならない要素, および (2) 繰返し $k = low : j - step$ の間で参照され, $k = j$ の繰返しを開始するときまで, 無効にならない要素の和として表現できる. すなわち, 次のように定義できる.

$$IN_h(j) = (IN_h^{low} - iterations(K_l', k, low : j - step : step)) \cup (iterations(OUT_l', k, low : j - step : step) - (\cup_{def} iterations(S_{def}, k, low : j - step : step))) \quad (11)$$

ここで, n_l は I_v^u の最終節点を表す. S_{def} は逆依存の定義側における配列要素の SAD を表す. $IN_v^u(low)$ は, $k = low$ すなわち, ループの最初の実行を開始するときの参照セクションを表す. この値は, I_v^u の要約節点 s の IN_s によって与えられる.

for $u := n$ **to** 0 **do**

区間グラフ G^u の各節点 n_i を前向きにたどりながら, データフロー方程式 (1)〜(3) に従って, IN_i, OUT_i を求める. その途中で区間 IN_v^u に出会ったときは, 次の処理を行う. IN_v^u のループ制御変数を k, k の初期値を low とする.

1. G^{u+1} における I_v^u の要約節点の IN_s を $IN_v^u(low)$ として, 繰返し $k = j$ における $IN_v^u(j)$ を求める.

2. $IN_v^u(j)$ を I_v^u のヘッダ節点 n_h の IN_h として, 区間内の節点を前向きにたどりながら, データフロー方程式 (1)〜(3) に従って, 各節点 n_i における IN_i, OUT_i を求める.

end

例 5. 例 4 の結果をもとに, 図 1 に用いられている配列 A に対して伝播の結果を考える. ここで, ループおよび要約節点を表す記号は例 4 と同じとする. n_{s1} には, 直接先行節がないので, $IN_1(low)$ は空集合となる. したがって, 式 (11) において, $IN_1(low) - iterations(K_l, I, 2 : j-1) = \phi$ となり, $IN_1(I)$ における D は $\langle A(1:99, 2:I-1), A(1:100, I-1)\rangle$ である.

L_1 のヘッダ節点では代入が行われないため, $OUT_1(I) = IN_1(I)$ となる. この結果をループ L_2 に対する要約節点 n_{s2} に伝播させる. L_1 に対する区間内の各節点における伝播フェーズの結果を**表 3** に示す.

L_3 には, 直接先行節における条件付き参照セクションはないので, 式 (11) に従って, $IN_{10}^{con}(I)$ は $\langle A(100, 2:I-1)\rangle$ である.

L_4 において, 直接先行節から伝播してきた $IN_{14}(low)$ は $A(1:99, 2:100)$, $A(1:100, 100)$ であり, 式 (11) に従って, $IN_{14}(I)$ は $\langle A(1:99, 2:100), A(1:100, 100), A(100, 30:I-1)\rangle$ である. □

6. 冗長なデータ転送の検出

本章では, データフロー解析の結果に基づいて, 冗長なデータ転送を検出する手法を示す.

6.1 冗長なデータ転送

ある配列への参照を考え, フローグラフの節点 n_i の入口における参照セクションの集合を $IN_i = \langle D_{IN}, M_{IN}\rangle$, 条件付き参照セクションの集合を $IN_i^{con} = \langle D_{IN}^{con}, M_{IN}^{con}\rangle$, 節点 n_i で参照されるセクションを $Gen_i = \langle D_i, M_i\rangle$ とする. このとき, $Gen_i \subseteq IN_i$ であれば, すなわち, 次の条件を満たせば, Gen_i はコピー参照可能であり, そのためのデータ転送は**完全冗長**である.

$D_i \subseteq D_{IN}$ **and** $M_i(D_i) \subseteq M_{IN}(D_{IN})$

$Gen_i \subseteq IN_i^{con}$ であれば，すなわち，次の条件を満たせば，Gen_i は条件付き参照可能であり，そのためのデータ転送は**条件付き冗長**である．

$D_i \subseteq D_{IN}^{con}$ **and** $M_i(D_i) \subseteq M_{IN}^{con}(D_{IN}^{con})$

続いて，部分冗長の検出について考える．部分冗長はコピー参照可能なセクションの一部が無効にされた場合に生じる．ある配列への参照を考え，節点 n_i の入口における参照セクションの集合を $IN_i = \langle D_{IN}, M_{IN} \rangle$，無効になるセクションの集合を $Kill_i = \langle D_{Kill}, M_{Kill} \rangle$，$n_i$ の出口にデータ転送のためのコード移動可能なセクションを $UPOUT_i = \langle D_{UPOUT}, M_{UPOUT} \rangle$ とする．D_{UPOUT} に含まれる1つのセクションを D_{UPOUT}^l $(l = 1, \ldots, m)$ で表す $(D_{UPOUT}^l \in D_{UPOUT})$．このとき，次の条件を満たせば，$UPOUT_i - Kill_i$ はコピー参照可能であり，そのためのデータ転送は**部分冗長**である．

$$D_{UPOUT}^l \subseteq D_{IN} \textbf{ and } D_{Kill} \subset D_{UPOUT}^l$$
$$M_{UPOUT}(D_{UPOUT}^l) \subseteq M_{IN}(D_{IN}) \textbf{ and}$$
$$M_{Kill}(D_{Kill}) \subset M_{UPOUT}(D_{UPOUT}^l)$$

$UPOUT_i$ は節点 n_i の出口でデータ転送のためのコード移動可能なセクションを表すので，$D_{UPOUT}^l \subseteq D_{IN}$ は，n_i の後続節点で参照されるセクションに対して，IN_i のデータを再利用することができることを意味する．$D_{Kill} \subset D_{UPOUT}^l$ は，$Kill_i$ によってコピー参照可能なセクションの一部を無効とすることを表す．

$UPIN_i$ と $UPOUT_i$ は方程式(7)および(8)を用いて求めることができる．その場合，IN_i を $UPOUT_i$，OUT_i を $UPIN_i$ に書き換え，フローグラフを最後の節点から最初の節点に向かって解析を進めるようにすれば，5章で述べた IN_i と OUT_i の求め方と同じである．

SPMD のコンパイルにおいては，それぞれの冗長性に対して，次のように冗長なデータ転送を削除する．完全冗長の場合，そのセクションに対応するデータ転送コードを完全に削除することができる．条件付き冗長の場合，その条件が成り立たないときに限って，データ転送を行うコードを生成する．部分冗長の場合，セクションの一部を無効する節点の出口で，その部分を転送するコードを生成する．

例 6. 図1のプログラムにおいて，配列 A のマッピング関数は例4に示したとおりなので，ここではデータ記述子だけを説明に用いる．**s6** において，Gen_6 は $A(1, I)$ であり，IN_6 は $A(1:99, 2:I)$，$A(1:100, I)$ である．$Gen_6 \subseteq IN_6$ なので，**s6** でのデータ転送は完全冗長であることが分かる．

s15 においては，Gen_{15} は $A(100, I)$ であり，IN_{15}^{con} $(= OUT_{L3}^{con})$ は $A(100, 2:100)$ であるから，$Gen_{15} \subseteq IN_{15}^{con}$ であり，**s15** でのデータ転送は条件付き冗長である．

s8 においては，IN_8 $(= OUT_{L1})$ は $A(1:99, 2:100)$，$A(1:100, 100)$ であり，$Kill_8$ は $A(1,2)$ であり，$UPOUT_8$ は $A(1, 2:100)$，$A(100, 30:50)$ である．$A(1, 2:100) \subseteq IN_8$，$Kill_8 \subset A(1, 2:100)$ であるから，$A(1, 3:100)$ は部分冗長である． □

6.2 実プロセッサ空間での冗長性の判定

IN_i は，SAD のリストとして表現されるために，仮想プロセッサ空間において $Gen_i \subseteq IN_i$ が成り立つかどうかを直接判定することが困難な場合がある．この関係の成否は，実プロセッサ上でのデータ集合の比較に置き換えると，簡単に判定できることがある．

まず，1次元配列について考える．節点 n_i での IN_i が $\langle A(0:n-1), \langle [1], [F_1(i) = a \times i + b] \rangle \rangle$ で与えられているとする．一般性を失わないように，$cyclic(k)$ とよばれるデータ配置ディレクティブを用いる．$cyclic(k)$ は，連続する k 個の仮想プロセッサを単位として，p 個の実プロセッサに周期的に配置することを意味する．$cyclic(\lceil n/p \rceil)$ と $cyclic(1)$ はそれぞれ HPF の **BLOCK** と **CYCLIC** に対応する．SAD $\langle D, M \rangle$ は，配列セクション D が仮想プロセッサ $M(D)$ に配置されることを表す．実プロセッサ上でのデータ集合の比較を行うために，この表現と $cyclic(k)$ とから，実プロセッサに配置される配列要素を求めるための方法を示す．すなわち，参照セクション $A(0:n-1)$ が $F_1(i) = a \times i + b$ によって仮想プロセッサにマッピングされ，さらに仮想プロセッサが $cyclic(k)$ によって p 個の実プロセッサに配置されるとき，プロセッサ m に配置される $A(0:n-1)$ 中の要素を求めることを考える．

図 4 に仮想プロセッサと実プロセッサ間の配置関係の例を示す．図4において，SAD は $\langle A(0:14), \langle [1], [F_1(i) = 3 \times i] \rangle \rangle$ としている．図中の数字は仮想プロセッサの番号を示す．配列要素がマッピングされている仮想プロセッサは下線で示す．このとき，仮想プロセッサ $a \times i + b$ (あるいは，配列要素 $A(i)$) が配置される実プロセッサの番号は，次の関数 $vpmap(i, p, k)$ によって与えられる．

$$vpmap(i, p, k) = ((a \times i + b) \bmod pk) \text{ div } k \quad (12)$$

ここで，パラメータ p，k およびマッピング関数 $F_1(i) = a \times i + b$ はコンパイル時に既知であり，$A(0:$

p_0					p_1					p_2				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44

図4 VPROCS(0:44) を3つのプロセッサへの配置 ($k = 5$)
Fig. 4 VPROCS(0:44) resides in 3 processors ($k = 5$).

$n-1$) の中で実プロセッサ m に配置される要素の番号 i は，$vpmap(i,p,k) = m$ $(0 \leq i \leq n-1)$ を満たす整数解として求めることができる．多次元配列に関しては，基本的に各次元について，ここで述べた方法を適用することによって，実プロセッサ m に配置される配列要素を求めることができる．その手法は Chatterjee らによって示されている[12]．

上述の方法によって，与えられた実プロセッサ m 上での IN_i と Gen_i の配列要素を，それぞれ新たな集合 IN_i^m と Gen_i^m で表す．

$$IN_i^m = \{A(i_1), A(i_2), A(i_3), \ldots\}$$
$$Gen_i^m = \{A(i'_1), A(i'_2), A(i'_3), \ldots\}$$

ここで，$i_1, i_2, \ldots$, $i'_1, i'_2, \ldots$ は定数であるから，$Gen_i^m \subseteq IN_i^m$ の成否の判定は，添字の比較によって簡単に実現することができる．

7. 提案手法の有効性評価

コピー参照セクションに基づく転送の最適化の効果を調べるために，分散メモリ型の並列マシン AP1000[7] 上で，3つのプログラムの実行を試みた．AP1000 は，1対1通信用のトーラスネットワーク（T-net），1対多通信用のブロードキャストネットワーク（B-net），およびバリア同期専用のネットワーク（S-net）の3種類の独立した通信ネットワークを持つ．実験に使用したシステムは256個の要素プロセッサから構成されている．そのうち，実験で用いたのは36台までである．個々のセルは，整数演算ユニット（IU），浮動小数点演算ユニット（FPU），16Mバイトのメインメモリからなる．IU と FPU は，128Kバイトのキャッシュに接続される．実験では，プログラム CKE (*Chemical Kinetics Equation*)[5]，GS (*Graphic Smoothing*)[15]，および SPEC92 ベンチマークプログラムの1つ SU2COR (*Quantum Physics*) を使用した．

CKE による評価実験では，化学動力方程式の解を求める計算のうちで，分子伝播式を計算する部分について，次の2種類の評価を行った．その部分のプログラムの骨子は図1に **s1** から **s7** までに示すとおりである．

(1) プロセッサ数 P を4に固定し，配列のサイズを変化させた場合の効果

配列のサイズ N を，それぞれ 200, 400, 600, 800 とする．配列 A, B, D の整列は図1に示すとおりである．このプログラムでは，**DISTRIBUTE VPROCS(BLOCK, BLOCK)** によってデータを実プロセッサに配置している．データ転送の最適化を行う場合と行わない場合とについて実行時間を測定した（**図5**）．

(2) 配列のサイズ N を600とし，プロセッサ数を変化させた場合の効果

プロセッサ数 P を1, 9, 16, 25, 36と変えながら，(1) の場合と同様に，冗長なデータ伝送を行う場合と行わない場合とについて実行時間を測定した（**図6**）．

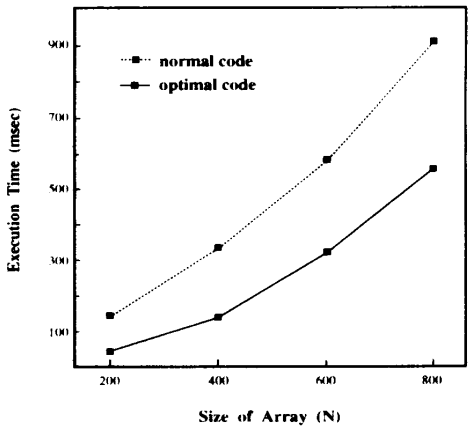

図5 配列のサイズを変化した場合の実行時間
Fig. 5 Execution time by varying the size of arrays.

一般に分散メモリマシンでは，メッセージ通信に要する時間はメッセージ転送の起動とメッセージの長さに依存する．CKE においては，前章で示したように **s6** におけるデータ転送が完全冗長である．**s6** の右辺の配列要素 $A(1, I)$ に対する転送はベクトル化することができるので，実際に削除されたメッセージ通信は1つだけである．したがって，この実験においては，データ転送のコストが単にメッセージの長さに依存することになっている．

GS についても，完全冗長性の削除による最適化の効果を測定した．プロセッサ数を 16, 32, 2つの

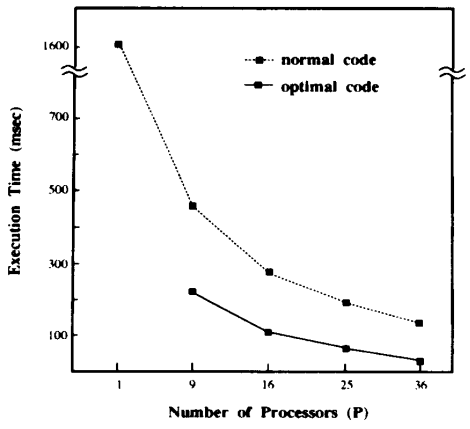

図6 プロセッサ数を変化した場合の実行時間
Fig. 6 Execution time by varying the number of processors.

表4 GS, SU2COR の実行時間の比較
Table 4 Speed up for GS and SU2COR.

Program	Array Size	Processors	Speed Up (%)
GS	400×400	16	15.2
		32	16.5
SU2COR	200×200	16	9.8
		32	11.1

配列のサイズを $S4(400,400)$, $S6(400,400)$ とし，**ALIGN (I,J) WITH VPROCS(I,J) :: S4, S6** および **DISTRIBUTE VPROCS(BLOCK,BLOCK)** によってデータを配置している．最適化を行わない場合の実行速度 (T_N) に対して，最適化を行った場合の実行速度 (T_O) の向上率を表4に示す．ここで，実行速度の向上率は $\frac{T_N - T_O}{T_N}$ によって求めたものである．

部分冗長性の削除による効果について，SU2COR を用いて計測を行った．SU2COR では，配列 $A(N,N)$ の全要素を参照した後，そのうちの N 個の要素の値が更新され，$A(N,N)$ の全要素が再参照される．N 個の更新された要素をベクトルで転送し，残り $N \times N - N$ 個の要素をそのまま利用することにした．これによって得られた実行速度の向上率を表4に示す．ここで，プロセッサ数を 16, 32，配列サイズを $A(200,200)$ とし，**ALIGN A(I,J) WITH VPROCS(I,J)** および **DISTRIBUTE VPROCS(BLOCK, BLOCK)** によってデータを配置している．

分散メモリマシンでは，メッセージ通信によるプロセッサ間データ転送のコストがかなり高く，コピー参照可能なセクションを求め，その情報を利用することによって，プログラムの実行時間を短縮できることを実験で示した．

8. 結　論

分散メモリ型のマシンでは，ハードウェアのピーク性能に対して，実際のプログラムで達成できる性能が低い，ということが指摘されている．その一因として，プロセッサ間でのデータ転送によるオーバヘッドが並列処理に大きく影響することがあげられる．本論文では，セクションアクセス記述子を導入し，データフロー方程式のパラメータと未知数はこの記述子で表される値をとるものとして，区間データフロー解析法によって，フローグラフの各節点の入口での参照セクションを求める手法を示した．最後に，その解析結果に基づく冗長なデータ転送の最適化を述べ，実験による効果を示した．この手法は具体的な通信方式に依存しないため，多くの並列化コンパイラへの応用が考えられる．

参考文献

1) Aho, A.V., Sethi, R. and Ullman, J.D.: *Compilers: Principles, Techniques, and Tools*, Addison-Wesley (1986).
2) Yoshida, A., Maeda, S., Ogata, W. and Kasahara, H.: A Data-Localization Scheme for Fortran Macro-Dataflow Compution, *Trans. IPSJ*, Vol.35, No.9, pp.1848–1860 (1994).
3) Callahan, D. and Kennedy, K.: Analysis of Interprocedural Side Effects in a Parallel Programming Environment, *Journal of Parallel and Distributed Computing*, No.5, pp.517–550 (1988).
4) Granston, E. and Veidenbaum, A.: Detecting Redundant Accesses to Array Data, *Proc. 1991 ACM International Conference on Supercomputing*, pp.854–869 (July 1991).
5) Rodrigue, G.: *Parallel Computations*, p.403, *Academic Press*, New York (1982).
6) High Performance Fortran Forum. High Performance Fortran Language Specification, Technical Report CRPC-TR92225, Rice University (Jan. 1993).
7) Ishihata, H., Horie, T., Inano, S., Shimizu, T. and Kato, S.: An Architecture of Highly Parallel Computer AP1000, *Proc. Pacific Rim Conference on Communications, Computers and Signal Processing*, pp.13–16 (May 1991).
8) Zima, H., Bast, H. and Gerndt, M.: SUPERB: A Tool for Semi-Automatic MIMD/SIMD Parallelization, *Parallel Comput.*, Vol.6, pp.1–18 (1988).
9) Miyoshi, I., Maeyama, K., Goto, S., Mori, S., Nakashima, H. and Tomita, S.: TINPAR:

A Parallelizing Compiler For Message-Passing Multiprosessors, *in Proc. IPSJ Joint Symposium on Parallel Processing*, pp.51–58 (May 1995).
10) Li, J. and Chen, M.: Compiling Communication-efficient Programs for Massively parallel machines, *IEEE Trans. Parallel and Distributed System*, Vol.2, No.3, pp.361–376 (1991).
11) Burke, M.: An Interval-based Approach to Exhaustive and Incremental Interprocedural Data-flow Analysis, *ACM Trans. Programming Languages and Systems*, Vol.12, No.3, pp.341–395 (1990).
12) Chatterjee, S., Gilbert, J., Long, F., Schreiber, R. and Teng, S.: Generating Local Addresses and Comunication Sets for Data-Parallel Programs, *Proc. ACM SIGPLAN '93 Symposium on Principle and Practice of Parallel Programming*, pp.149–158 (May 1993).
13) Hiranandani, S., Kennedy, K. and Tseng, C.: Evaluation of Compiler Optimizations for Fortran D on MIMD Distributed-Memory Machines, *Proc. 1992 ACM International Conference on Supercomputing*, pp.1–14 (July 1992).
14) Gross, T. and Steenkiste, P.: Structured Dataflow Analysis for Arrays and its use in an Optimizing Compiler, *Software-Practice and Experience*, Vol.20, No.2, pp.133–155 (1990).
15) Gardner, W.E.: Machine-aided Image Analysis, p.342, *The Institute of Physics*, Bristol and London (1978).
16) Li, X. and Harada, K.: Removing Redundant Synchronization from Nested Parallel Loops, *Proc. 1993 International Conference on Computer Science*, pp.25–28 (July 1993).
17) Li, X. and Harada, K.: An Optimal Mapping of a Global Array to Hierarchical Memory Systems with Three Levels, *Proc. 1994 IEEE International Symposium on Parallel Architectures, Algorithms and Networks*, pp.67–74 (Dec. 1994).
18) Li, X. and Harada, K.: An Optimization for Data Transmissions on Distributed Memories by Data Flow Analysis, *Proc. 1995 IPSJ Joint Symposium on Parallel Processing*, pp.43–50 (May 1995).
19) Li, X. and Harada, K.: A Framework for Removing Redundant Data Transmission on Distributed Memories, *Proc. 1995 International Conference on Parallel Computing*, pp.28–34, North-Holland (Sep. 1995).

(平成7年9月4日受付)
(平成8年3月12日採録)

李 暁傑（正会員）

1986年中国北京工業大学計算機科学科卒業．同年中国北京計算機科学技術研究所に入所．1986～1990年同研究所ソフトウェア工学研究室助手，研究員．1993年慶應義塾大学大学院修士課程修了（計算機科学専攻）．現在，同大学大学院後期博士課程在籍．並列分散処理，プログラミング言語および並列化コンパイラに興味を持つ．電子情報通信学会会員．

原田 賢一（正会員）

1940年生．1966年慶應義塾大学大学院修士課程修了（管理工学専攻）．1967年同大学工学部助手．1970～1989年同大学情報科学研究所助手，専任講師，助教授，教授．1989年4月より同大学理工学部計測工学科教授．同大学大学院計算機科学専攻教授兼任．この間，1973～1975年米国メリーランド大学訪問研究員．工学博士．ソフトウェア工学，プログラミング言語およびその処理系の研究に従事．ACM，IEEE，ソフトウェア科学会会員．