ハイパフォーマンス
コンピューティング 57−16
（1995. 8. 25）

分散メモリにおけるHPF配列要素の有効性解析

李 暁傑　　原田 賢一

慶應義塾大学 理工学部
横浜市港北区日吉 3-14-1

概　要

分散メモリマシン上でのプログラムにおいて配列を扱う場合，全体をいくつかに分割し，各部分をそれぞれのプロセッサのメモリに配置する方法が用いられている．分散メモリマシンはグローバルなアドレス空間をもたないため，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，プロセッサ間の通信によってアクセスを実現する必要がある．プログラムの実行効率を向上させるための工夫の１つとして，この転送操作の削減があげられる．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ったとき，その要素の値がそれ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，データ転送を避けることができる．本稿は，この点に着目し，High Performance Fortranで書かれたプログラムを対象として，分散メモリにおけるHPF配列の有効性を，データフロー解析法の１つである区間解析法によって見つけ出す手法を提案する．

Availibility of HPF Array Elements on Distributed Memories

Xiaojie LI　　Ken'ichi HARADA

Department of Computer Science, Keio University
3-14-1, Hiyoshi, Kouhoku-ku, Yokohama, 223, Japan

Abstract

Reducing the overhead of data transmissions is crucial to harnessing the potential of distributed memory multiprocessors. In typical numerical applications, a significant portion of data transmissions arises from accesses to blocks of array elements or regions. Data transmission overheads due to such traffic can be reduced by using compile-time information to detect when data can be reused. This paper presents a data flow analysis framework for determining the availability of data on a virtual processor grid. The data availability information obtained is useful for determing whether an array section can be reused and optimizing communication for distributed address space multiprocessors.

1 はじめに

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，効率のよ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．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の並行プログラムに対しては，これまでに Fortran D，High Performance Fortran などのコンパイラが開発されている．それらのコンパイラでは，**Owner computes rule** [4] 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．右辺の評価において，それを実行するプロセッサのメモリにないデータの参照に対しては，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を行なう際に，各プロセッサ上で実行されるプログラムを考慮し，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．以下，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 で無条件**コピー参照可能**，あるいは無条件**有効**という．コピー後，n までの実行において x の値が更新されない条件分岐があれば，x は P_i の点 n で**条件付きでコピー参照可能**，あるいは**条件付きで有効**という．

あるプロセッサが所有しているデータの更新に伴って，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，右辺に現われる配列要素 x 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，x 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，x の再転送を避けることができる．分散メモリマシンによる大規模な分散環境においては，プロセッサ間でのデータ通信の最適化のために，データの有効と無効に関する**有効性**を求めるのは有用である．

本研究では，SPMD によるプログラム実行において，プログラムの各計算点におけるコピー参照可能なデータの集合を求めるために，**アクセス** (代入・参照) されるデータとそれらの動作を行うプロセッサとを，グローバルな範囲で解析する手法を示す．その解析アルゴリズムには，**区間解析法** (*interval analysis*) [7] [11] を用いる．

2 対象モデルとデータフロー方程式

2.1 解析対象のプログラムとデータの有効性

本研究では，High Performance Fortran(**HPF**) [3] 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セクションを求めることを考える．

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

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
```

図 1: プログラム例 (一部)

例 1. 図 1 のプログラムを用いて，本稿で述べるデータの有効性の例を示す．s3 の実行において，*owner computes rule* 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づけられている．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 (有効) であることがわかる． □

ループ中で実行される代入文について，ループの制御変数の値によって決まる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，セクション アクセス記述子とよぶ表現法を導入する．

セクション アクセス記述子 (*section access descriptor*,SAD) は，配列要素または配列要素の集まりを表す記述子 (**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 からなり，$< D, M >$ で表す．D に M を適応すること，すなわち $M(D)$ によって，それらのプロセッサが得られる．SAD に関する記述は文献 [13] を参照されたい．

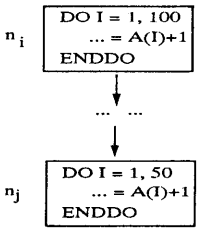
2.2 データフロー方程式

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．解法については 3 章で述べる．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，各配列の使用を独立

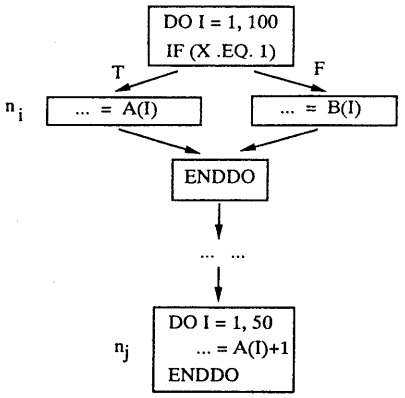
に扱い，それらに対するセクション アクセス情報を集合として扱えばよい．

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．

参照セクションは無条件参照セクションと条件つき参照セクションとに分類される．無条件参照セクションとはある節点で参照され，その後無条件で n に到達し，n においてコピー参照できるセクションである．これに対して，条件つき参照セクションは条件分岐によって参照されたセクションである．このようなセクションは，条件を満たさない場合，n に到達することは保証されない．本稿では，この 2 種類のデータ有効性をそれぞれに求めておく．以下，無条件参照セクションをたんに参照セクションとよぶ．



(a) 無条件参照セクション A(1:100)



(b) 条件付き参照セクション A(1:100)

図 2: 参照セクションと条件つき参照セクション

まず，プロセッサを固定し，その上で実行されるプログラムのフローグラフについて，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示す．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 (条件付きで有効な) 参照セクションの集合
- OUT_i (OUT_i^{CON})：n_i の出口における (条件付きで有効な) 参照セクションの集合
- K_i (K_i^{CON})：n_i の出口における (条件付きで) 無効セクションの集合
- Gen_i：n_i 中の文の実行によって，参照されるセクションの集合
- Gen_i^{CON}：n_i が条件分岐であり，n_i 中の文の実行によって，参照されるセクションの集合
- $Kill_i$：n_i 中の文の実行によって，無効になるセクションの集合
- $Kill_i^{CON}$：n_i が条件分岐であり，n_i 中の文の実行によって，無効になるセクションの集合

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]．

$$IN_i = \cap_p OUT_p, \tag{1}$$

$$K_i = (\cup_p K_p) \cup Kill_i, \tag{2}$$

$$OUT_i = (IN_i - (Kill_i \cup Kill_i^{CON})) \cup Gen_i. \tag{3}$$

と

$$IN_i^{CON} = \cup_p OUT_p^{CON}, \tag{4}$$

$$K_i^{CON} = (\cup_p K_p^{CON}) \cup Kill_i^{CON}, \tag{5}$$

$$OUT_i^{CON} = (IN_i^{CON} - Kill_i^{CON}) \cup Gen_i^{CON}. \tag{6}$$

の 2 組で表す．ここで，p は n_i の直接先行節である．Gen_i，Gen_i^{CON}，$Kill_i$，および $Kill_i^{CON}$ は既知の定数集合として，方程式から変数集合の IN_i，IN_i^{CON}，OUT_i，および OUT_i^{CON} を求める．コピー参照可能なセクションを求めるためには，ループ中で実行される代入文について，ループの制御変数の値によって決まる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，上に示した集合を SAD 記述子で表す．

3 コピー参照可能なセクションを求めるための区間解析法

ここでは，区間解析法 (*interval analysis*)[7] [11] をもとに，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．IN_i^{CON}，OUT_i^{CON} の求め方は IN_i，OUT_i の求め方と同様であるため，後者だけを示す．

区間解析法は，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．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．ループは 1 つの区間を構成する．その場合，ループの最後の文に当る節点を**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．最終節点に集められるセクション アクセス情報は，逆順辺を経て再びヘッダ節点に達することになる．以下では，簡約可能グラフ (*reducible graph*) を仮定して，前章で述べたデータフロー方程式をもとに各フェーズにおける処理内容を述べる．

3.1 簡約フェーズ

簡約フェーズの各ステップでは，与えられたフローグラフを区間分割し，各区間を**要約節点** (*summary node*) とよぶ 1 つの節点で表す．そして，もとのグラフの節点間の辺を区間どうしの辺で表した高次のフローグラフを作成する．グラフ全体が単一の節点になるまでこのステップを繰り返す．

簡約ステップでは，各区間について，そこ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' を求めていく．1 つの区間についての簡約が終了したら，最終節点における K_i' と OUT_i' を，それぞれその区間の要約節点の Gen_s，$Kill_s$ とする．

ループを構成する 1 つの区間 I_h について，ローカルなセクション アクセス情報の求め方を示す．そのヘッダ節点 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，次の初期設定を行う．

$$K_h' = \phi, \quad OUT_h' = \phi$$

このあと，I_h 内の各節点 n_i を前向きに走査し，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．その最終節点を n_l とすると，K_l' と OUT_l' には，区間内でのアクセス情報を表す式が得られる．

OUT_l' と K_l' を，I_h の要約節点 n_s の Gen_s，$Kill_s$ に与える際，I_h 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 の制御パラメタでの表現 (範囲表現) に置き換える．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 $expand(S, k, low : high : step)$ を定義する．

拡張関数 $expand(S, k, low : high : step)$ 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．

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

この変換に伴って，この添字から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変換する．上に示した式が S のセクション記述子の中で t 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 次元目の位置を与える．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．

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

区間 I_h 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 の制御パラメタを $low : high : step$ とする．

$$Gen_s = expand(OUT_l', k, low : high : step) - (\cup_{def} expand(S_{def}, k, low : high : step)) \quad (7)$$

$$Kill_s = expand(K_l', k, low : high : step) \quad (8)$$

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．`Doall` に対応する区間であれば，逆依存は存在しないので，その Gen_s は $expand(OUT_l', k, low : high : step)$ となる．

例 1. 図 1 のプログラム例に対する簡約の結果を表 1 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 によって構成されるループをそれぞれ L_1，L_2 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} で表す．

配列 A の要素についての簡約の結果は次のとおりである．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，$OUT_2'(= Gen_{s2})$ 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) $expand(OUT_6', I, 2:100)$，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 1 に示す Gen_{s1} の結果が得られる． □

表 1: 図 1 のプログラム例に対する簡約結果

Step	SAD set	D
Elimination Step 1	OUT'_3	$A(J,I)$
	K'_3	$B(I,J)$
	Gen_{s2}	$A(1:100,I)$
	$Kill_{s2}$	$B(I,1:100)$
Elimination Step 2	OUT'_5	$A(1:100,I)$
	K'_5	$A(100,I-1)$ $B(I,1:100)$
	OUT'_6	$A(1:100,I)$
	K'_6	$D(I)$ $A(100,I-1)$ $B(I,1:100)$
	Gen_{s1}	$A(1:99,2:100)$ $A(1:100,100)$
	$Kill_{s1}$	$D(2:100)$ $A(100,1:99)$ $B(2:100,1:100)$

3.2 伝播フェーズ

伝播フェーズでは，高次のフローグラフから低次のフローグラフへ向かって，要約節点に集めた情報を，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ていく．この操作の繰返しにより，もとのフローグラフの各節点 n_i について，参照セクションの集合 IN_i，および OUT_i が求められる．

伝播フェーズのあるステップを考え，n_h をヘッダ節点とする区間を I_h とし，I_h に対する要約節点を n_s とする．また，そのループ制御変数を k，k の制御パラメタを $low, high, step$ とする．まず，n_s の入口での参照セクション IN_s を求める．n_s の直接先行節を p とすると，それらのセクションは，$IN_s = \cap_p OUT_p$ によって得られる．IN_s は，I_h にとって，繰返しを開始する時点，すなわち $k = low$ のときの参照セクションである．それを IN_h^{low} で表す．

ループが k によって制御されるとき，ヘッダ節点を含めて区間内の各節点については，k のある繰返しでの参照セクションを表す IN_i を求めたい．まず，$k = j$ となったときのヘッダ節点での参照セクション $IN_h(j)$ を求めることを考える．

ループ制御変数 (k) が値 low から出発して，j となったときのヘッダ節点 h に対する $IN_h(j)$ は，次の要素の和として表現できる．

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
2. 繰返し $k = low : j - step$ の間で参照され，$k = j$ の繰返しを始めるときまで，無効にならない要素

つまり次の式によって表現できる．

$$IN_h(j) = (IN_h^{low} - expand(K'_l, k, low : j - step : step))\cup(expand(OUT'_l, k, low : j - step : step) - (\cup_{def} expand(S_{def}, k, low : j - step : step))) \quad (9)$$

ここで，n_l は I_h 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．このとき，IN_h は $IN_h(j)$ によって与えられる式中の j をループ制御変数 k で置き換えたもの，すなわち $IN_h(k)$ として得られる．

このあとは，前章で示したデータフロー方程式 (1) と (3) を用いて，各節点 n_i について IN_i，OUT_i を求めていけばよい．ただし，節点 n_i が要約節点のときは，その節点が表す区間 I_i に対して，ここで述べた伝播フェーズを再帰的に適用する．I_i がループ制御変数 l によって制御され，l が $step$ きざみで，終値 $high$ をとるものとする．また，I_i のヘッダ節点を $n_{h'}$ とする．このとき，l によるループが終了したときの参照セクションの集合，すなわち OUT_i は次の式によって与えられる．

$$OUT_i = IN_{h'}(high + step)$$

例 2. 例 1. の結果をもとに，図 1 に示したプログラム例を再び考える．ループ L_1，L_2 を表す区間の要約節点を n_{s1}，n_{s2} とする．n_{s1} には，直接先行節がないので，IN_{s1}，すなわち IN_1^{low} は空集合となる．したがって，式 (9) において，

$$IN_1^{low} - expand(K_l, I, 2 : j - 1) = \phi$$

となり，$IN_1(j)$ は次の式によって与えられる．

$$IN_1(j) = expand(OUT'_6, I, 2 : j - 1) - (\cup_{def} expand(S_{def}, I, 2 : j - 1)) =< A(1:100, 2 : j - 1) - A(100, 1 : j - 2) >$$

表 2: 図 1 のプログラムに対する伝播の結果

SAD set	Equation	D
IN_2	$OUT_1(I) = IN_1(I)$	$A(1:99,2:I-1), A(1:100,I-1)$
OUT_2	$(IN_2 - Kill_2)\cup Gen_2$	$A(1:99,2:I-1), A(1:100,I-1:I)$
IN_5	OUT_2	$A(1:99,2:I-1), A(1:100,I-1:I)$
OUT_5	$(IN_5 - Kill_5)\cup Gen_5$	$A(1:99,2:I), A(1:100,I)$
IN_6	OUT_5	$A(1:99,2:I), A(1:100,I)$
OUT_6	$(IN_6 - Kill_6)\cup Gen_6$	$A(1:99,2:I), A(1:100,I)$

与えられたプログラムに対して，ある配列への参照を考え，節点 n_i の入口におけるその参照セクションの集合を IN_i=< D_{IN}, M_{IN} >，節点 n_i で参照されるセクションを Gen_i=< D_i, M_i > 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冗長である．

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

4 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ー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，まず，配列要素へのアクセスとその動作を行うプロセッサとを併せて表現するためにセクション アクセス記述子を導入した．そして，データフロー方程式のパラメ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．解析の結果に基づいて，データ転送を最適化することができる．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] A. V. Aho, R. Sethi, and J. D. Ullman : *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986.

[2] E. Granston and A. Veidenbaum, *"Detecting Redundant Accesses to Array Data,"* in *Proc. 1991 ACM Int. Conf. Supercomputing*, pp.854-869, July 1991.

[3] *High Performance Fortran Forum. High Performance Fortran Language Specification*, Technical Report *CRPC-TR92225*, Rice University, Jan. 1993.

[4] H. Zima, H. Bast, and M. Gerndt, *" SUPERB : A Tool for Semi-Automatic MIMD/SIMD parallelization,"* Parallel Comput., Vol.6 (1988), pp.1-18.

[5] I. Miyoshi, K. Maeyama, S. Goto, S. Mori, H. Nakashima, and S. Tomita, *"TINPAR: A Parallelizing Compiler For Message-Passing Multiprosessors,"* in *Proc. of Joint Symposium JSPP'95*, pp.51-58, May 1995.

[6] J. Li and M. Chen, *"Compiling Communication-efficient Programs for Massively parallel machines,"* IEEE Trans. Parallel and Distributed System, Vol. 2, No. 3 (1991), pp.361-376.

[7] M. Burke, *An Interval-based Approach to Exhaustive and Incremental Interprocedural Data-flow Analysis," ACM Trans. Programming Languages and Systems*, Vol.12, No.3 (1990), pp.341-395.

[8] S. Hiranandani, K. Kennedy, and C. Tseng, *"Evaluation of Compiler Optimizations for Fortran D on MIMD Distributed-Memory Machines,"* in *Proc. 1992 ACM Int. Conf. Supercomputing*, pp.1-14, July 1992.

[9] S. Pande and K. Psarris, *"A Compilation Technique for Varying Communication Cost NUMA Architectures,"* in *Proc. of Parallel Architectures and Languages Europe*, pp.49-60, July 1994.

[10] T. Doi, K. Hayashi, and T. Shindo, *"Code Generation Using Stride Data Transfer Mechanism," IPSJ SIG Notes*, HPC-52-1, pp.1-6, July 1994.

[11] T. Gross and P. Steenkiste, *"Structured Dataflow Analysis for Arrays and its use in an Optimizing Compiler,"* Software - Practice and Experience, Vol.20, No.2 (1990), pp.133-155.

[12] X. Li and K. Harada, *"An Optimal Mapping of a Global Array to Hierarchical Memory Systems with Three Levels,"* in *Proc. of International Symposium ISPAN'94*, pp.67-74, Dec. 1994.

[13] X. Li and K. Harada, *"An Optimization for Data Transmissions on Distributed Memories by Data Flow Analysis,"* in *Proc. of Joint Symposium JSPP'95*, pp.43-50, May 1995.

[14] X. Li and K. Harada, *"A Framework for Removing Redundant Data Transmissions from HPF Programs,"* in *Proc. of 1995 International Conference on Parallel Computing*, Sep. 1995.