FLoPS：分散メモリ型並列計算機を対象とした並列化コンパイラ

進藤達也† 岩下英俊† 土肥実久†
萩原純一† 金城ショーン†

本論文では，分散メモリ型並列計算機を対象とした並列化コンパイラ FLoPS について述べる．FLoPS は，分散メモリ型並列計算機向けの標準言語案 High Performance Fortran (HPF) と，富士通で開発された VPP Fortran の両方を入力言語としてコンパイルすることができ，AP1000 用のコードを生成する．FLoPS には2つの重要な特徴がある．1つは，VPP Fortran の記述を用いたマシン独立な最適化である．もう1つは，AP1000 のハードウェアによってサポートされたダイレクトリモートデータアクセスを用いたコード生成法である．AP1000 を用いた実験結果によって，FLoPS における最適化とコード生成の効果を示す．

FLoPS: A Parallelizing Compiler for Distributed Memory Parallel Computers

TATSUYA SHINDO,† HIDETOSHI IWASHITA,† TSUNEHISA DOI,†
JUNICHI HAGIWARA† and SHAUN Y. KANESHIRO†

This paper presents the design and implementation of FLoPS – a parallelizing compiler for distributed memory parallel computers. FLoPS compiles both HPF and VPP Fortran programs as its input and generates parallelized code for the AP1000. There are two important features implemented in FLoPS. The first is machine-independent optimizations based on VPP Fortran language features. The second is a code generation and optimization technique using direct remote data access (DRDA) mechanism supported by the AP1000 hardware. Based on experiments performed on the AP1000, this paper shows the effects of the new code generation and optimization techniques.

1. はじめに

分散メモリ型並列計算機を対象とした言語やそのコンパイラの研究開発が数多く行われている[2),5),7),12)~17)]．これらはコンパイラ技術によりグローバルアドレス空間を提供しており，プログラマはプロセッサ間通信の詳細やプロセッサ間に分散されたデータのローカルアドレスを考慮することなく並列プログラミングを行うことができる．グローバルアドレス空間上の配列の添字から各プロセッサのローカルメモリ上の配列の添字への変換や，通信同期のためのコードの生成は，コンパイラによって行われる．また，分散メモリ型並列計算機のためのコードとしては，通信の最適化をいかに行うかが大きく性能に影響する．このような最適化も，コンパイラにとって重要な課題である．我々は，分散メモリ型並列計算機を対象とし，入力言語として High Performance Fortran (HPF)[4)] と VPP Fortran[11)] を扱う並列化コンパイラ FLoPS (Fujitsu Laboratories' Optimizing and Parallelizing System) を開発しており，今回 AP1000[6),8),10)] を対象とした処理系を実現した．HPF は，分散メモリ型並列計算機のための標準化案として提案されている並列処理言語であり，VPP Fortran は，富士通によって開発された言語である．本論文では，FLoPS における最適化技術を中間表現形式と関連させて紹介し，また AP1000 の特徴的な通信機構であるダイレクトリモートデータアクセス (DRDA) を用いたコード生成法について述べる．

関連する研究としては，分散メモリ型並列計算機上で，グローバルアドレスを実現するコンパイラに関する研究が多数成されている[2),5),7),12)~17)]．最近では，HPF のコンパイラも報告されており[1),14)]，Applied Parallel Research の xhpf や Pacific-Sierra Research の VAST/HPF や Portland Group の HPF コンパイラ[9)]などいくつかの製品も出ている．Split-C[2)]では，

† 富士通株式会社
Fujitsu Ltd.

List 1 DO ループ表現

```
1   (DO  do-attribute  [partition-info]
2       (DOVAR var) (LB expr) (UB expr) (STEP expr)
3     [(PREBARRIER)][(POSTBARRIER)][(SYNCVAR  var)]
4     (BODY
5       statements
6     )
7   )
```

DRDA と同様の方式として Active Message[3] と呼ぶ直接リモートプロセッサのメモリにアクセスする方式が採用されている．しかしながら，データ転送の一括化に関しては，Split-C では，Bulk Data Operation と呼ぶ連続データの転送に関するものを扱うのみで，ストライドデータ転送の有用性や利用法については言及していない．

2. FLoPS における中間表現

中間表現（IM）フォーマットは，はS式の形式をしており，Fortran コードの情報に加え，HPF を表現するための分散や並列化の情報を持つ．さらに，VPP Fortran の記述から派生した，マシン独立な最適化を表現するための記法を持つ．本章では最適化のための表現を中心に述べる．以下で，IM フォーマットの例において，大文字はキーワード，小文字はパラメータを表すものとする．

2.1 DO ループ

SPMD 型の並列化の対象になる DO ループは，IM フォーマット上で **List 1** のように表せる．ここで，var は変数名を expr は式を表す．[ ] で囲まれた情報は必要なときのみ記述される．ここで，do-attribute は，SEQ, PARA, SPDO, SMOVE のどれかがマークされる．SEQ は逐次ループを，PARA は並列化可能ループを，SPDO は並列化可能ループの中から実際に並列コード生成の対象とするもの（SPREAD DO）を示す．SMOVE は VPP Fortran における一括転送を指示する SPREAD MOVE を表す．SPREAD MOVE は，代入文のみがボディに記述可能であり，その代入がコード生成時に DRDA を用いた一括転送に置き換えられる．partition-info は，SPREAD DO における計算分散を表す．partition-info については 2.2 節で述べる．PREBARRIER と POSTBARRIER は，それぞれ DO ループの前後のバリア挿入を示す．SPREAD DO はデフォルトとして両方のバリアが指定され，最適化により移動あるいは削除がされる．SYNCVAR は，SPREAD MOVE のときのみに有効で，データ転送の終了待ちを行うための同期用変数を指定する．

List 2 分割記述表現 (partition-info)

```
1   (partition-name
2     (DIST          (dist-type [blocksize])
3                  ... (dist-type [blocksize]))
4     (OVERLAP   (lowersize   uppersize)
5                ... (lowersize   uppersize))
6     (PROCESSORS  processor-name)
7     (MAPPING       mapping-func)
8   )
```

2.2 分割記述（partition-info）

分割記述（**List 2**）は，配列データに対するデータ分割と並列 DO ループに対する計算分割の両方に使われる．HPF の template と align ディレクティブの組合せに対応する．DIST は，各次元の分散を表し dist-type として，BLOCK，CYCLIC がマッピング先のプロセッサの次元数分指定される．OVERLAP は，各次元のオーバラップエリアのサイズを表す．lowersize は下限側のサイズを表し，uppersize は上限側のサイズを表す．オーバラップエリアを持たない次元は，(0 0) となる．OVERLAP は，VPP Fortran で陽に指定された場合，あるいは，HPF コードの最適化結果として生成された場合に指定される．PROCESSORS では，分割される対象をマッピングするプロセッサ名を指定する．マッピング関数（mapping-func）は，配列データや DO ループのインデクス $\vec{i}$ からテンプレートのインデクス $\vec{t}$ への写像を表す関数 f である．

$$\begin{aligned}\vec{t} &= f(\vec{i}) \\ &= A\vec{i} + \vec{b}\end{aligned}$$

$$\begin{pmatrix} t_1 \\ \vdots \\ t_n \end{pmatrix} = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \begin{pmatrix} i_1 \\ \vdots \\ i_m \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

ここで，$\vec{t}$ はプロセッサ次元数 n 個の要素からなるベクトル，$\vec{i}$ は，m 個の要素からなるベクトルである．A は，どの行あるいは列においても，非零要素はたかだか1つであるような整数行列であり，$\vec{b}$ は n 個の要素からなる整数ベクトルである．以降，A を係数部と

呼び，$\vec{b}$ をオフセット部と呼ぶ．IM フォーマットでは，mapping-func 部に，この式と等価な情報を持つ．

2.3 アクセス属性

IM フォーマットでは変数のアクセスについて，アクセス属性としてそれがつねにローカルメモリへのアクセス（self）なのか，リモートメモリへ通信をともなったアクセスの可能性がある（remote）のかを示すフラグが存在する．IM フォーマットにおいて，配列アクセスを表す式は以下のように表現する．

```
(ARRAY (SYM access-attribute var)
       (SUBSCRIPTS expr ... expr))
```

ここで，var は配列名，expr は各次元の添字を表す式であり，access-attribute に self か remote が示される．access-attribute が remote であるアクセスは，通信最適化の対象となる．最適化が適用できない場合は，その場に通信のコードが生成される．

マッピング関数を用いて self の定義を行う．配列データのインデクス $\vec{d}$ からテンプレートのインデクス $\vec{t}$ へのマッピングを表す関数を $\vec{t} = f(\vec{d})$ とし，DO ループのインデクス $\vec{c}$ からテンプレートのインデクス $\vec{t}$ へのマッピングを表す関数を $\vec{t} = g(\vec{c})$ とする．また，DO ループのインデクス $\vec{c}$ からアクセスされる配列データのインデクス $\vec{d}$ へのマッピングが，マッピング関数で表すことができる場合を self 判定の条件とし，その関数を $\vec{d} = h(\vec{c})$ とする．オーバラップエリアを考慮しない場合の，self の定義は以下のようになる．

$$g(\vec{c}) \equiv f(h(\vec{c}))$$

オーバラップエリアの各次元のサイズを要素としたベクトルとして，下限側を $\vec{l}$ とし上限側を $\vec{u}$ とすると，オーバラップエリアを考慮する場合の，self の定義は以下のようになる．

$$\vec{l} \leq f(h(\vec{c})) - g(\vec{c}) \leq \vec{u}$$

List 3 において，2 行目の DO は SPREAD DO であり，その分割は，配列 A，B，C の 1 次元目にアラインされているとすると，5 行目のステートメントは，**List 4** のように IM で表され，配列 A と配列 C のアクセス属性は self になり，配列 B のアクセス属性は remote になる．

2.4 Overlapfix と movewait

オーバラップエリア[5]を最新の値で更新するための指示を IM 内では，overlapfix 命令として以下のように表現する．

```
(OVERLAPFIX (var ... var)
            [(SYNCVAR var)])
```

List 3 Matmul (matrix multiplication).

```
1    !HPF$ DISTRIBUTE(BLOCK,*):: A, B, C
2          DO 214 I=1, L        ! SPDO
3            DO 215 K=1, N      ! SEQ
4              DO 216 J=1, M    ! SEQ
5                C(I,K)=C(I,K)+A(I,J)*B(J,K)
6     216      CONTINUE
7     215    CONTINUE
8     214  CONTINUE
```

List 4 IM フォーマットにおけるアクセス属性

```
1    (STORE tREAL4
2      (ARRAY (SYM SELF tREAL4 c)
3        (SUBSCRIPTS (SYM SELF tINT4 i)
4          (SYM SELF tINT4 k)))
5     (+ tREAL4
6       (ARRAY (SYM SELF tREAL4 c)
7         (SUBSCRIPTS (SYM SELF tINT4 i)
8           (SYM SELF tINT4 k)))
9       (* tREAL4
10        (ARRAY (SYM SELF tREAL4 a)
11          (SUBSCRIPTS (SYM SELF tINT4 i)
12            (SYM SELF tINT4 j)))
13        (ARRAY (SYM REMOTE tREAL4 b)
14          (SUBSCRIPTS (SYM SELF tINT4 j)
15            (SYM SELF tINT4 k))))))
```

var でオーバラップエリアのデータ更新の対象になる配列名を指定する．また，SYNCVAR に続く var では，データ更新のためのデータ転送の終了待ちを行うための同期用の変数を指定する．

SPREAD MOVE あるいは overlapfix のためのデータ転送の終了を待つための指示を IM 内では，movewait 命令として以下のように表現する．

```
(MOVEWAIT (SYNCVAR var))
```

var には，対応する SPREAD MOVE あるいは，overlapfix で指定された同期のための変数を指定する．

3. 汎用最適化

3.1 計算分割

計算分割では，並列 DO ループの各イテレーションとそれを実行するプロセッサの対応を決定する．高い性能を得るためには，並列 DO ループのボディに表れる配列のアクセスができるだけ通信を必要とせずに実行できるように，計算分割が決定されなければならない．我々が採用した多数決法は，並列 DO ループのボディに表れる各配列アクセスにとって好ましい計算分割を求め，その中で多数を占める分割に決定する．そのアルゴリズムを次に示す．

(1) do-attribute として “PARA” のマークされた DO ループのボディ内の各配列アクセスに対し

て最適な計算分割 g を $g(\vec{c}) = f(h(\vec{c}))$ により求める．

(2) 同一配列変数に対して，同一の添字で行うアクセスが，ボディ内に複数回表れる場合は，そのうち 1 つのみを記録する．これは，まったく同一のアクセスに対しては，1 つの SPREAD MOVE しか生成しないため，通信コストを 1 アクセス分として見積もるためである．

(3) 各計算分割 g の式において，係数の部分だけを比べオフセットの部分を無視して多数決を行う．これは，オフセットが異なっていても係数が一致していれば，オーバラップエリアを生成することでローカル化できる可能性があるからである．

(4) 最後に，係数が一致しているものの中から，オフセットの値を多数決で求め，並列 DO ループの計算分割を決定する．

3.2 ローカル化とオーバラップエリア生成

ローカル化では，与えられたデータ分割と計算分割と各アクセスにおける添字の式から，そのアクセスがつねにローカルなデータに対するものかどうかを調べる．つねにローカルであるアクセスに対しては，アクセス属性として self を記し，それ以外のものは remote を記す．この判定は，2.3 節における “self の定義” より行う．

次に，オーバラップエリアを考慮する．アクセス対象としている配列に，すでにオーバラップエリアが宣言されている場合は，2.3 節における “オーバラップエリアを考慮した self の定義” よりアクセス属性を判定する．一方，アクセス対象としている配列に，オーバラップが宣言されていない場合は，マッピング関数が以下のように表せる場合には，$\vec{w}$ サイズのオーバラップエリアを宣言することで，アクセス属性を self とすることができる．ここで，$\vec{w}$ は定数値のみからなるベクトルとする．

$$f(h(\vec{c})) - g(\vec{c}) = \vec{w}$$

オーバラップエリアを宣言する際に，$\vec{w}$ の各要素において正の値を持つものが上限側のオーバラップエリアを表し，負の値を持つものが下限側のオーバラップエリアを表す．各絶対値がそれぞれの次元のオーバラップエリアのサイズである．コンパイラあるいはユーザによる制限以内であれば，そのサイズのオーバラップエリアを生成する．なお，オーバラップエリアを考慮してアクセス属性を “self” としたアクセスが DO ループ内に存在する場合には，オーバラップエリア上のデータを更新するために overlapfix 命令をその DO ループの直前に挿入する．

List 5 SPREAD MOVE 生成前の並列 DO ループ

```
1     DO I=L, U              ! PARA or SPDO
2        ... = A(aI+b)       ! remote attribute
3        B(cI+d) = ...       ! remote attribute
4     ENDDO
```

List 6 SPREAD MOVE 生成後の並列 DO ループ

```
1     DO I=L, U              ! SPMV
2        LA(I) = A(aI+b)     ! global to local
3     ENDDO
4     DO I=L, U              ! PARA or SPDO
5        ... = LA(I)         ! self attribute
6        LB(I) = ...         ! self attribute
7     ENDDO
8     DO I=L, U              ! SPMV
9        B(cI+d) = LB(I)     ! local to global
10    ENDDO
```

3.3 SPREAD MOVE 生成

属性として，SPDO あるいは PARA がマークされた並列 DO ループを対象に，SPREAD MOVE 生成を行う．並列 DO ループのボディに表れる配列アクセスのうち remote がマークされたもの，すなわち通信を必要とするアクセスを，SPREAD MOVE で一括転送に置き換え，並列 DO ループの前後に生成する．**List 5** に，並列 DO ループの例を示し，**List 6** にその並列 DO ループに対して，SPREAD MOVE 生成を施した結果を示す．List 6において，配列 A に対するテンポラリ配列として LA を，配列 B に対するテンポラリ配列として LB をアクセス属性が self となるように割り付ける．その結果，並列 DO ループにおけるデータの参照は，ローカル領域に対するものとなり，通信は発生しない（List 6 の 4 行目から 7 行目）．

3.4 アクセス属性を用いたバリア最適化

FLoPS において，プロセッサ間の同期は主にバリアを用いている．バリアは，HPF や VPP Fortran が扱うデータパラレル型のプログラミングモデルに十分な同期機構であり，さらに，AP1000 ではバリヤを高速実行する機構がハードウェアに設けられている．ここでは，アクセス属性を利用したバリアの最適化として，外側ループへのバリア移動と同一プロセッサ内のデータ依存に対するバリア削除を紹介する．

バリア移動は，ネストした DO ループの内側に SPREAD DO がある場合に，SPREAD DO の前後で呼ばれるバリアを，より外側の DO ループの前後に移動することによりバリア同期のオーバヘッドを削減する最適化である．バリア移動が適用可能かどうか

List 7 リモートアクセスをともなったSPREAD DO

```
1       DO I=1, N                ! SEQ
2         BARRIER
3         DO J=L, N              ! SPDO
4           A(I,J) = ...         ! self attribute
5           ... = f(A(I-1,J))    ! remote attribute
6         ENDDO
7         BARRIER
8       ENDDO
```

List 8 リモートアクセスのないSPREAD DO

```
1       DO J=1, N                ! SEQ
2         BARRIER
3         DO I=L, N              ! SPDO
4           A(I,J) = ...         ! self attribute
5           ... = f(A(I,J-1))    ! self attribute
6         ENDDO
7         BARRIER
8       ENDDO
```

List 9 リモートアクセスのないSPREAD DOにおけるバリア移動

```
1       BARRIER
2       DO J=1, N                ! SEQ
3         DO I=L, N              ! SPDO
4           A(I,J) = ...         ! self attribute
5           ... = f(A(I,J-1))    ! self attribute
6         ENDDO
7       ENDDO
8       BARRIER
```

の条件判定は，データ依存を持つ配列アクセスの属性を調べることで行える．すべての依存を持ったアクセスの属性がselfであれば，バリア移動が可能である．逐次ループで囲まれたSPREAD DOの例を**List 7**と**List 8**に示す．ここで，配列Aは1次元目に沿ってプロセッサ間に分割されているものとする．どちらも，5行の参照と4行目の定義がデータ依存を持っている．List 7では，参照がremoteであるためバリア移動は適用できない．一方，List 8では，参照も定義もselfであり，**List 9**に示すようにバリア移動が適用できる．

バリア削除では，DOループ間での不要なバリアを削除する．DOループの前後に挿入されたバリアは，一般にはデータ依存がない場合に削除できる．さらに，データ依存がある場合においても，その依存を起こしている2カ所のアクセスのアクセス属性がともにselfならば，プログラムの実行順序が保証されるためバリアは削除できる．**List 10**の例では，2つのDOループネスト間で配列Aへのアクセスによるデータ依存

List 10 selfアクセスを用いたバリア削除

```
1       BARRIER
2       DO J=1, N                ! SEQ
3         DO I=L, N              ! SPDO
4           A(I,J) = ...         ! self attribute
5         ENDDO  ENDDO
6       ENDDO  ENDDO
7       BARRIER                  ! can be eliminated
8       DO J=1, N                ! SEQ
9         DO I=L, N              ! SPDO
10          ... = f(A(I,J-1))   ! self attribute
11        ENDDO
12      ENDDO
13      BARRIER
```

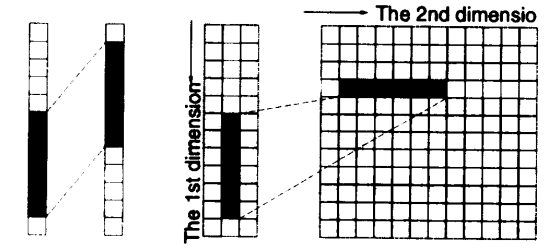


図1 メモリ間のデータ転送
Fig. 1 Data transfers among memory regions.

が存在する．しかしながら，どちらのアクセスもアクセス属性がselfであるならば，7行目のバリアを削除することができる．

4. コード生成

4.1 AP1000におけるストライドをサポートしたダイレクトリモートデータアクセス

ダイレクトリモートデータアクセス（DRDA）は，リモートプロセッサ上のメモリに対して，直接データの書き込みあるいは読み出しを行うメカニズムである[6]．DRDAを用いたコードジェネレーションにおいて，データ転送を効率良く行うためのストライドデータ転送が重要である．まず，データ転送のハードウェアが連続領域のみを扱う場合を考える．転送対象データが，ローカル側とリモート側の両者におけるメモリ上で連続領域に割り当てられている場合（図1(a)）には，その全体に対するアクセスを1回のデータ転送として実現することができる．しかしながら，図1(b)のような場合は，2次元方向アクセスする側で一定間隔ごとのとびとびの領域をアドレッシングする必要があり連続領域とならない．その結果，一括転送は行えず，転送回数の増加にともないデータ転送起動時間のオーバヘッドも増加する．ストライドデータ転送は，

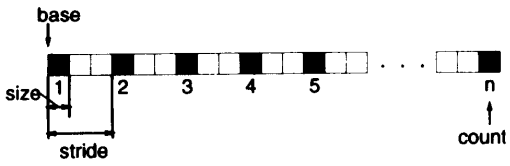
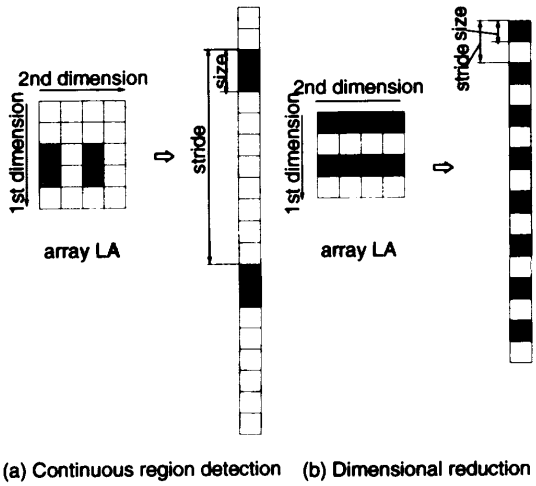


図2 AP1000でサポートするストライド転送
Fig. 2 Stride transfer supported on the AP1000.

List 11 ストライド転送をサポートしたDRDAインタフェース

```
1    remoteRead (pid,r_base,l_base,
2                r_stride,l_stride,
3                size,count)
4    remoteWrite(pid,r_base,l_base,
5                r_stride,l_stride,
6                size,count)
```



(a) Continuous region detection (b) Dimensional reduction

図3 DRDA生成のための最適化
Fig. 3 Optimization techniques for DRDA generation.

このようなとびとびの領域に対するデータ転送を，1回の起動で実行するものである．

AP1000では，図2に示す1次元のストライドデータ転送をDirect Memory Access（DMA）機構により，ハードウェアでサポートしている．ストライド転送をサポートしたDRDAのインタフェースをList 11に示す．List 11において，pidは通信相手のリモートプロセッサのIDを表す．r_base，r_strideはリモートプロセッサ側のメモリにおける図2のbaseとstrideに対応し，l_base，l_strideはローカルメモリにおける図2のbaseとstrideに対応する．また，List 11におけるsizeとcountは，リモートとローカル共通で，図2のsizeとcountに対応する．

4.2 ストライドを用いたコード生成と最適化

一般に，配列はプロセッサ間にまたがって分散され，1つのSPREAD MOVEで複数のプロセッサと通信が必要になる．コードジェネレーションでは，グローバルアドレス空間を対象に書かれたSPREAD MOVEをAP1000のDRDAライブラリを用いて複数のプロセッサのローカルメモリに対するアクセスに変換する．

まず，SPREAD MOVEは通信相手のプロセッサごとのDOループに変換される．次に，その最内側DOループをプロセッサと通信するDRDAのランタイムライブラリに置き換える．SPREAD MOVEの本体にある代入文が，リモートプロセッサのメモリから，ローカルメモリへの代入であればremoteReadを用い，ローカルメモリからリモートプロセッサのメモリへの代入であればremoteWriteを用いる．

我々は，マシン依存の最適化として，2次元以上のSPREAD MOVEをAP1000でサポートする1次元のDRDAにマッピングできるようリニアライズし，通信回数を減らす方法を2種類開発した．1つは，continuous region detection法である．これは以下の条件を満たす場合（図3(a)）に，配列の1次元方向のアクセスと別のもう1つの次元方向へのアクセスを1つのDRDA呼び出しに置き換えるものである．

- 配列の1次元方向のアクセスが連続領域（ストライドなし）である．
- 別の次元方向へのアクセスが存在し，その添字の上下限が固定である．

もう1つは，dimensional reduction法である．これは，以下の条件を満たす場合（図3(b)）に，n次元目のアクセスと$n+1$次元目のアクセスを1つのDRDA呼び出しに置き換えるものである．

- 配列のある次元nの大きさがストライドの大きさでちょうど割りきれる．
- 配列の次元$n+1$方向のアクセスが連続であり，その添字の上下限が固定である．

5. 実験結果

カーネルプログラムとして，List 3のMatmulとLivermoreベンチマークのkernel7を用いて，通信最適化個々の効果を調べた．また，より大きなプログラムとして，SPECfpよりswm256とtomcatvを用いて，バリア最適化を含めた総合的な効果を調べた．各プログラムは，データ分散の記述をHPFのディレクティブで与え，DOループの並列化はデータ依存解析を用いて自動並列化した．tomcatvでは，自動化の対象にならなかったDOループを人手で組込関数に書き換えたコードを用いた．速度向上率（speedup ratio）としては，逐次版のプログラムを1台のプロセッサで実行した速度を基準にした．

Matmulの測定結果を図4に，kernel7の測定結果

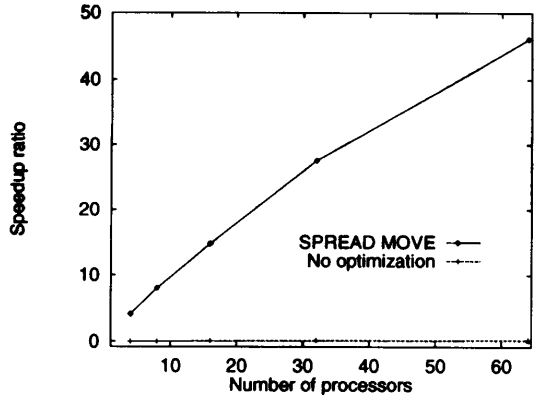


図4 AP1000 における Matmul の性能
Fig. 4 Performance of Matmul on the AP1000.

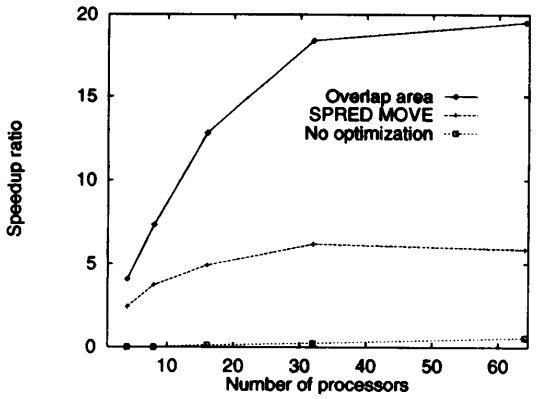


図5 AP1000 における kernel7 の性能.
Fig. 5 Performance of kernel7 on the AP1000.

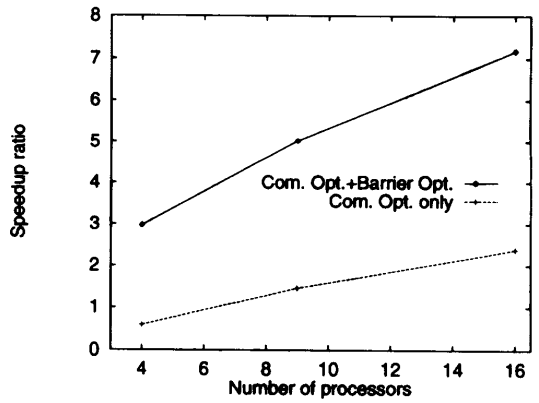


図6 AP1000 における swm256 の性能.
Fig. 6 Performance of swm256 on the AP1000.

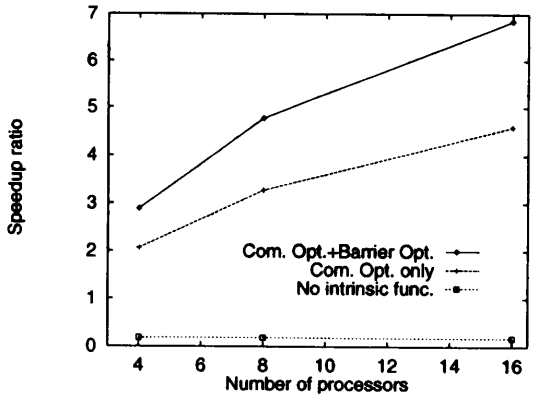


図7 AP1000 における tomcatv の性能
Fig. 7 Performance of tomcatv on the AP1000.

を図5に示す．それぞれ，SPREAD MOVE の生成と overlap area の生成により性能が向上しており，どちらも最適化を行わなかった場合は，通信のオーバヘッドにより，逐次版よりも性能が大きく低下している．

swm256 の測定結果を図6に示す．swm256 の通信最適化としては，SPREAD MOVE 生成が適用された．このグラフでは，通信最適化だけを行ったものに比べ，さらにバリア最適化を行うことで高い性能が得られることを示している．swm256 では，2重にネストした DO-loop が多く，その内側が並列化されるケースがある．このようなループネストに対してバリア移動が最適化として適用できた．また，連続する2つのループにおいては，バリア削除が適用できた．

tomcatv の測定結果を図7に示す．通信最適化として，オーバラップエリア生成と SPREAD MOVE 生成が適用された．また，同期の最適化の効果も確認できた．参考までに組込関数で書き換えなかったコードに対してすべての最適化を適用した結果も，No intrinsic func. としてグラフに示す．

ストライド転送を用いたコード生成の効果は，tomcatv において確認ができた．コード生成にストライド転送を用いた場合と用いなかった場合を比較し，表1に示す．tomcatv のオーバラップエリアは，2次元配列の2次元目に沿って割り付けられるため，そのエリアのデータ転送を一括転送で行うためにストライド転送が必要になる．ストライドを用いたコードジェネレーションを行うことによって，行わない場合に比べて，平均データ長は長くなり，データ転送数が小さくなっている．また，これにともない実行時間が短縮されている．

表1 ストライドデータ転送の効果

Code generation	Average data size	Data transfer count	Execution time (sec)
No stride	8.0 words	1645832	35.95
Stride	129.5 words	101640	22.89

6. おわりに

HPF と VPP Fortran を扱うコンパイラ FLoPS について述べた．その中間表現はマシン非依存の最適化をサポートするために，VPP Fortran をベースに設

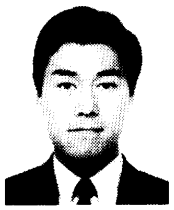
計されている．AP1000上の通信プリミティブとしては，ダイレクトリモートデータアクセスを用いた．ストライドデータ転送は，ダイレクトリモートデータアクセスを用いて効果的にコード生成するために重要である．AP1000を用いた実験結果により，FLoPSにおける最適化とコード生成の効果を示した．

謝辞 本研究に対しご支援，ご指導いただいております石井光雄氏，河内実氏，白石博氏，神谷幸男氏，池坂守夫氏に感謝いたします．

参考文献

1) Bozkus, Z., Choudhary, A., Fox, G., Haupt, T. and Ranka, S.: Fortran 90D/HPF Compiler for Distributed Memory MIMD Computers: Design, Implementation, and Performance Results, *Supercomputing'93*, pp.351–360 (1993).
2) Culler, D.E, Dusseau, A., Krishnamurthy, S.C.G.A., Lumetta, S., von Eicken, T. and Yelick., K.: Parallel Programming in Split-C, *Supercomputing'93*, pp.262–273 (1993).
3) Eicken, T., Culler, D.E, Goldstein, S.C and Schauser, K.E.: Active Messages: a Mechanism for Integrated Communication and Computation, *The 19th Annual International Symposium on Computer Architecture*, pp.256–266 (1992).
4) Forum, H.P.F.: High Performance Fortran Language Specification Version 1.0 (1993).
5) Gerndt, H.M and Zima, H.P.: Optimizing Communication in SUPERB, *CONPAR90-VAPP IV*, pp.300–311 (1990).
6) Hayashi, K., Doi, T., Horie, T., Koyanagi, Y., Shiraki, O., Imamura, N., Shimizu, T., Ishihata, H. and Shindo, T.: AP1000+: Architectural Support of PUT/GET Interface for Parallelizing Compiler, *Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.196–207 (1994).
7) Hiranandani, S., Kennedy, K. and Tseng, C.: Compiler optimizations for Fortran D on MIMD Distributed-Memory Machines, *Supercomputing '91*, pp.86–100 (1991).
8) Horie, T., Ishihata, H., Shimizu, T., Kato, S., Inano, S. and Ikesaka, M.: AP1000 Architecture and Performance of LU Decomposition, *1991 International Conference of Parallel Processing Vol.I*, pp.634–635 (1991).
9) II, R.B., Choudhary, A., Meadows, L., Nakamoto, S. and Schuster, V.: Retargetable High Performance Fortran Compiler Challenges, *CompCon'93*, pp.137–146 (1993).
10) Ishihata, H., Horie, T., Inano, S., Shimizu, T. and Kato, S.: An Architecture of Highly Parallel Computer AP1000, *IEEE Pacific Rim Conf. on Communications, Computers, and Signal Processing*, pp.13–16 (1991).
11) Iwashita, H., Okada, S., Nakanishi, M., Shindo, T. and Nagakkura, H.: VPP Fortran and Parallel Programming on the VPP500 Supercomputer, *ISPAN'94* (for the poster session), pp.165–172 (1994).
12) Koelbel, C. and Mehrotra, P.: Compiling global Name-Space Parallel Loops for Distributed Execution, *IEEE Trans. Parallel and Distributed Systems*, pp.440–451 (1991).
13) Li, J. and Chen, M.: Compiling Communication-Efficient Programs for Massively Parallel Machines, *IEEE Trans. Parallel and Distributed Systems*, pp.361–376 (1991).
14) Nakatani, T.: Compiling HPF for A Cluster of Workstations, 並列処理シンポジウム JSPP'93, pp.1–6 (1993).
15) Ruhl, R. and Annaratone, M.: Parallelization of FORTRAN Code on Distributed-Memory Parallel Processors, *International Conference on SUPERCOMPUTING '90*, pp.342–353 (1990).
16) Sabot, G. and Wholey, S.: CMAX: A Fortran Translator for the Connection Machine System, *International Conference on SUPERCOUMPUTING '93*, pp.147–156 (1993).
17) Zima, H., Brezany, P., Chapman, B., Mehrota, P. and Schwald, A.: Vienna Fortran－A Language Specification Version 1.1, Acpc/r 92-4 (1992).

(平成7年8月30日受付)
(平成8年9月12日採録)



進藤 達也（正会員）

1983年早稲田大学理工学部電子通信学科卒業．1983年より（株）富士通研究所．1990～1992年スタンフォード大学客員研究員．1994年より富士通（株）．CAD専用マシン，並列処理の研究に従事．1986年，篠原記念学術奨励賞．

岩下　英俊（正会員）

1986年愛媛大学工学部電子工学科卒業．1988年同大学院工学研究科修士課程修了．同年（株）富士通研究所入社．1994年より富士通（株）．並列Fortran言語の設計，並列化コンパイラの研究開発に従事．電子情報通信学会会員．

土肥　実久（正会員）

1988年大阪府立大学工学部電気工学科卒業．1990年同大学院工学研究科電気工学専攻博士前期課程修了．同年（株）富士通研究所入社．1994年より富士通（株）．機械翻訳，CADシステム，並列コンパイラの研究に従事．情報処理学会会員．

萩原　純一（正会員）

1991年早稲田大学理工学部電子通信学科卒業．1993年同大学院修士課程修了．同年（株）富士通研究所入社．1994年より富士通（株）に勤務．並列処理システム，並列化コンパイラの研究に従事．情報処理学会，IEEE，ACM各会員．

金城ショーン

1991年マサチューセッツ工科大学Department of Electrical Engineering and Computer Science卒業．1993年同大学院修士課程修了．1993～1995年MIT-Japan Internship Programにより（株）富士通研究所，富士通（株）に勤務．並列処理システム，並列化コンパイラの研究に従事．Tau Beta Pi member.