ハイパフォーマンス
コンピューティング 48－2
（１９９３．８．１９）

AP1000 を対象とした VPP Fortran 処理系の実現と評価

進藤達也　岩下英俊　土肥実久　萩原純一[1]
(株)富士通研究所　並列処理研究センター[2]

VPP Fortran はグローバルなアドレス空間をプログラマに提供した並列プログラミング用言語である．本言語は本来，富士通製スーパーコンピュータVPP500を対象に開発された．今回我々は，分散メモリ型並列計算機 AP1000 を対象にした VPP Fortran の言語処理系 Fortran/AP を実現した．実現にあたって新しい通信方式，ダイレクトリモートデータアクセス法を採用した．この方式によりFortran/AP はインデクス配列による間接アクセスを効果的に実行することができる．本稿ではその方式のインプリメント法について述べ，AP1000を用いた実験結果を示す．

An implementation and evaluation of a VPP Fortran compiler for AP1000

Tatsuya SHINDO, Hidetoshi IWASHITA, Tsunehisa DOI, Junichi HAGIWARA
Parallel Computing Research Center, FUJITSU LABORATORIES LTD.

VPP Fortran is a parallel programing language which supports a global address space for programmers. It was originally designed for VPP500 which is a supercomputer developed by Fujitsu. We implemented Fortran/AP which is a VPP Fortran compiler for AP1000 with a new data communication method, direct remote data access. With the method, Fortran/AP handles indirect data accesses with an index array efficiently. We show a technique to implement the method and the experimental results with AP1000.

[1]E-mail:{shindo, hideto, micky, jhagiw}@flab.fujitsu.co.jp
[2]211 川崎市中原区上小田中1015

1 はじめに

分散メモリ型並列計算機はそのスケーラビリティから，大規模問題を解くための次世代スーパーコンピュータの候補の１つと考えられる．このような並列計算機が広く使われるためには，プログラマにとってハードウェアの詳細（例えば，不連続なアドレス空間，メッセージパッシングによるプロセッサ間通信 等）を意識せずにプログラミングが行なえることが重要である．特に，プログラマに対してグローバルアドレス空間をサポートすることは，プログラミングの容易化や既存ソフトェア資産の継承という観点から有用である[4][5][6][7][8]．

本稿では，グローバルなアドレス空間をサポートした言語VPP Fortran の AP1000 を対象にした処理系について述べる．VPP Fortran は，FORTRAN77に並列化とデータ分散に関するディレクティブを追加した言語である．以下では，言語そのものを VPP Fortran と呼び，AP1000 を対象にした VPP Fortran の処理系を Fortran/AP と呼ぶ．

従来，このような言語のインプリメンテーションには，send と receive を基本にしたメッセージパッシングによる通信をコンパイル時にスケジュールする方式がとられてきた[6][7][8]．しかしながら，配列の添字に配列が表れるような変数のアクセスは，どのプロセッサがデータを send し，どのプロセッサがそのデータを receiveするかといった対応をコンパイル時にスケジュールする事が一般的には不可能であり，ブロードキャスティング等のオーバーヘッドを伴う．我々は，Fortran/AP を実装するにあたり，異なるプロセッサの特定のアドレスに対して直接データを書き込んだり，また異なるプロセッサの特定のアドレスから直接データを読み出す方式(ダイレクトリモートデータアクセス)を採用した．この方式の採用により，コンパイル時にsendとreceiveの対応をとらねばならないといった制約を避ける事ができる．

以下，2章では，VPP Fortran の言語の紹介を行なう．3章では，グローバルアドレスを想定したプログラムを実行するために必要なデータ通信方式について述べる．4章では，異なるプロセッサの特定のアドレスに対して直接データをアクセスする方式をAP1000[10]上に実現する方法について述べる．5章では，Fortran/AP を用いたプログラムの性能評価について述べる．6章では，Fortran/APと関連する研究について紹介し，その関連を述べる．

2 VPP Fortran

2.1 言語概要

VPP Fortran は，FORTRAN77を拡張した並列処理用言語であり，富士通製の並列ベクトル計算機VPP500用のFORTRAN[1]として開発された．FORTRAN77に，並列化とデータ分散に関する情報を与えるディレクティブを追加することにより実現されている．プログラマは逐次計算機用にプログラムを書くのと同様に，グローバルなアドレス空間を用いてプログラミングが行え，各プロセッサに分散されたメモリ内のローカルなアドレスやプロセッサ間の通信の詳細を意識する必要は無い．ディレクティブはコメントの形態をしており，正しく挿入されていれば，逐次マシン上でディレクティブを単なるコメントとして無視して実行した結果とVPP Fortran として並列計算機上で実行した結果は一致する．この特徴により，逐次マシン用に書かれたプログラムから並列化プログラムへの連続的な書き替えを容易にしている．

例として配列乗算のプログラムの一部をlist1に示す．list1において，!XOCLで始まる行がディレクティブである．1行目はNPROC台構成のプロセッサ集合Pを宣言している．2,3,4行目で配列Aと配列Bのデータ分散法を定義している．“/(P)”と書かれた次元はその次元の要素をプロセッサ集合Pに分散させる事を意味する．“：”と書かれた次元は分散しない．ここで，GLOBAL宣言は，配列の任意の要素に任意のプロセッサからアクセスされる可能性があることを示している．LOCAL宣言は配列のどの要素もそれを保持するプロセッサ(owner)からしかアクセスされないことを示している．5行目と13行目で，最外側のDOループをプロセッサ集合Pに分散して並列実行することが示されている．

2.2 プログラムチューニング用言語機能

VPP Fortran では，プログラムをチューニングするために複数のデータの一括転送を指定するディレクティブを用意している．分散メモリ型の並列計算機において，データ転送の処理時間は転送の回数に大きく依存するために，一括転送がプログラムの高速化のために重要だからである．

一括転送のためのディレクティブには，SPREAD MOVE と OVERLAP FIX がある．SPREAD MOVE は，配列から配列への複数要素の転送を一括化するための記述である．DO ループで書かれた配列間代入に対して，

List 1 VPP Fortranによる配列乗算の例

```
1    !XOCL PROCESSOR P(NPROC)
2    !XOCL LOCAL  AL(/(P),:)
3    !XOCL GLOBAL B(:,/(P))
4    !XOCL LOCAL  CL(/(P),:)
5    !XOCL SPREAD DO /(P)
6          DO 214 I=1, L
7            DO 215 K=1, N
8              DO 216 J=1, M, 4
9                CL(I,K)=CL(I,K)+AL(I,J)*B(J,K)
10    216      CONTINUE
11    215    CONTINUE
12    214  CONTINUE
13   !XOCL END SPREAD
```

List 2 データ転送の一括化をした配列乗算の例

```
1            K1=(N/NPROC)*(IDRPROC()-1)+1
2            DO 214 K=1,N
3    !XOCL SPREAD MOVE
4              DO 217 J=1,M
5                TMPB(J)=B(J,K1)
6      217     CONTINUE
7    !XOCL END SPREAD (X)
8    !XOCL MOVEWAIT (X)
9    !XOCL SPREAD DO /(P)
10             DO 215 I=1,L
11               DO 216 J=1,M
12                 CL(I,K1)=CL(I,K1)+AL(I,J)*TMPB(J)
13     216       CONTINUE
14     215     CONTINUE
15   !XOCL END SPREAD
16             K1=MOD(K1,N)+1
17     214   CONTINUE
```

SPREAD MOVE のディレクティブを挿入することでコンパイラに指示する．OVERLAP FIX は，配列をプロセッサにブロック分散した際に，となり合うプロセッサで境界部分のデータを参照しあう場合のチューニングに用いる．まず，互いに参照する境界部分のコピーをそれぞれのプロセッサに持たせる．このコピー部分を最新の値に更新する指示が OVERLAP FIX である．

SPREAD MOVE を用いてlist1に書かれた乗算プログラムにチューニングを施した例をlist2に示す．list1の例では配列Bについてループ内でアクセスの度に通信を行なっているが，一方，list2の例では，乗算を行なう前に配列Bのデータのうち必要なものをローカルな配列TMPBにSPREAD MOVE で一括転送している（3〜8行目）．1行目に使われている関数IDVPROC()は，実行しているプロセッサのID（1以上の値）を得るためのものである．

3 プロセッサ間のデータ通信方式

本節ではまず，従来メッセージパッシングマシン用のコード生成で使われてきた「スケジュールされた send & receive」とその短所について述べる．次に，AP1000 を対象にしたVPP Fortran 処理系(Fortran/AP) の実現で用いた通信方式「ダイレクトリモートデータアクセス」について述べる．

3.1 スケジュールされた send & receive によるコード生成

分散されたデータに対するアクセスを実現するために，send系のプリミティブとreceive系のプリミティブを対にして用いるコードを生成する方式が既に採用されている[5][6][7][8]．この方式では，分散されたあるデータがプロセッサ間にまたがってreadされる場合には，そのデータの持ち主(owner)であるプロセッサはそのデータの参照を行なうプロセッサにそのデータをsendし，そのデータのreadを行なうプロセッサは送られたデータをreceiveし参照する．プロセッサ間にまたがってwriteする場合も同様である．

このように，send と receive を用いて通信をスケジュールする方式は，send と receive の対が実行時にならないと決定できない場合に，効率の良いコード生成ができないといった短所を持つ．このような状況は，プロセッサ間に分散された配列のアクセスの際に，添字として配列の要素が使われる場合に起こりうる(list3)．

このケースでは，IDXAやIDXBの内容により，どのプロセッサがどの要素をどのプロセッサに送るかが決定される．なお，IDXAやIDXB

List 3 実行時にsendとreceiveのペアが決定する例

```
1    !XOCL SPREAD DO /PA1
2          DO 100 I=1,256
3             ...=A(IDXA(I))
4             B(IDXB(I))= ...
5     100  CONTINUE
6    !XOCL END SPREAD
```

はプロセッサに分散されているか，あるいは全領域をプロセッサ毎に持つが内容はそれぞれ異なるような場合を想定する．この場合，コンパイル時に send と receive の対応をとることができない．そのため，一般には転送対象となるデータがどのプロセッサに存在するか分からないため，全対全通信が必要になる．

3.2 ダイレクトリモートデータアクセス

Fortran/AP の実現では，プロセッサ間にまたがるデータ通信のインターフェイスとして，特定のプロセッサ上のメモリに対して直接データを書き込み読みだしをおこなうダイレクトリモートデータアクセス法を採用した．具体的には，以下のようなインターフェイスを持ったランタイムルーチンを用意した．

```
readRemote(pid,r_adrs,l_adrs,size)
writeRemote(pid,r_adrs,l_adrs,size)
```

pidはプロセッサID, r_adrs はリモートマシンの先頭アドレス, l_adrs はローカルマシンの先頭アドレス, size は転送するデータのサイズの指定である．readRemote は，pidで指定されたプロセッサの r_adrs から size 分のデータを読みだし自プロセッサの l_adrs から size 分の領域に読み込む．writeRemote は，自プロセッサの l_adrs から size 分のデータを，pidで指定されたプロセッサの r_adrs から size 分の領域に書き込む．

これにより，コンパイル時に send と receive の対応をとる必要はなくなり，実行時に転送すべきデータが決定する前記のようなコードに対しても実際に必要な通信のみで済むようになる．

4 ダイレクトリモートデータアクセス の AP1000上での実現

異なるノードプロセッサ上にあるデータに直接アクセスする機能は，ある種の並列計算機ではハードウェアで実現されている[1][2][9]．本節では，そのような機構を持たないメッセージパッシング型のマシン上にダイレクトリモートデータアクセスを実現する方法について述べる．

4.1 Send & receive を用いたダイレクトリモートデータアクセスの実現法

以下の2通りの実現方式を検討した．

- メッセージ到着に対する割り込みハンドラにより実現する方式．
- メッセージパッシングライブラリを用いてエミュレーションする方式．

前者は，あるプロセッサから send されたメッセージが他のプロセッサに届いた際に起動される割り込みハンドラを変更する事で実現する．本来のメッセージパッシングのライブラリでは，この割り込みハンドラは届いたメッセージを受信用バッファに格納する．ダイレクトリモートデータアクセスを実現するためには，到着したメッセージに対して割り込みハンドラ内で必要な処理，たとえばwriteRemote に対する処理 とか readRemote に対する処理を行なうように変更する．このような割り込みルーチン内で到着したメッセージに対する処理を行なう方式はactive messages [3] として提案されている．

後者が，今回新たに試みた方式である．前者のように並列計算機のOSや割り込みハンドラレベルでの改造は行なわず，既存のメッセージパッシング用のライブラリを用いて実現する．writeRemote や readRemote の要求はメッセージパッシングライブラリの sendにより，アクセス相手のプロセッサに送られる．しかし，要求はアクセス相手の受信バッファに単に格納されるだけである．そこで，その要求を読みだし対応する処理を実行する仕組みをアクセス相手のプロセッサに用意せねばならない．これを実現するために以下のようなランタイムライブラリを実現した．

- 受信バッファ内に他のプロセッサからの要求が届いているかを確認し，あればその処理を行なうプロシージャ systemMessageHandler を用意する．
- 通信と同期に関するライブラリ (readRemote, writeRemote, send, receive, グローバル演算，バリア同期) が呼ばれたときに，その内部から最低1回 systemMessageHandler を呼ぶようにする．

- readRemote や receive でのデータ待ちやバリア同期での待ち状態の間には，待ちの状態の監視と systemMessageHandler の呼びだしを交互に行なうようにする．

これにより，readRemote, writeRemote の要求は次のステップで処理される．以下でプロセッサAは要求する側で，プロセッサBは要求を受ける側とする．

1. プロセッサAは，プロセッサBに対して要求に対応するメッセージをsendする．
2. 要求を表すメッセージがプロセッサBの受信バッファに格納される．
3. プロセッサBが通信あるいは同期のライブラリを呼ぶか，あるいは既に待ちの状態になっている．
4. プロセッサBで systemMessageHandler が呼ばれ，メッセージバッファ内の要求がとりだされ対応する処理(readRemoteの場合はデータが読み出されパケットに詰められる．writeRemoteの場合は送られたデータを指定されたアドレスに書き込む．）が実行される．プロセッサAに対して要求に対する返答（readeRemoteの場合は読み出したデータ．writeRemoteの場合は単なる処理終了の知らせ．）をsendする．
5. 返答を表すメッセージがプロセッサAの受信バッファに格納される．
6. プロセッサAが通信あるいは同期のライブラリを呼ぶか，あるいは既に待ちの状態になっている．
7. プロセッサAで systemMessageHandler が呼ばれ，返答に対する処理（次節で述べる同期のためのカウンタ操作．readeRemoteの場合は読み出されたデータを該当するアドレスに書き込む．）を行なう．

4.2 通信処理終了の検出

ここでは，並列 DO ループ(SPREAD DO, SPREAD MOVE)の終了時に，全てのプロセッサにおいて必要な通信が終ったことを検出する事の重要性とその実現方法を述べる．

スケジュールされた send & receive を用いて生成されたコードと違い，ダイレクトリモートアクセスを用いて生成されたコードでは，あるプロセッサに対するアクセス要求は，そのプロセッサの処理とは非同期で行なわれる．そこで，全てのプロセッサにおいてその並列DOループ内で行なうべき通信処理が終了していることをなんらかの方法で検出してから，次の処理に移らねばならない．この手続きを踏まないと，データが読み出される前に新しいデータで上書きされてしまったり，逆に読み出すべきデータが用意される前に古いデータを読んでしまう事態が発生する．

通信処理終了の検出には，各プロセッサ毎に持つカウンタの操作とバリア同期により実現する．カウンタは，他のプロセッサに remoteRead の要求を出している数を数える readRequestCounter と，それに対する返答の数を数える readReplyCounter, 他のプロセッサに writeRemote の要求を出している数を数える writeRequestCounter と，それに対する返答の数を数える writeReplyCounter の4種類を用意する．具体的には以下のステップを並列DOループで実行し，通信処理終了を検出する．

1. 並列DOループの担当するイテレーションの実行を行なう．
2. readRequestCounter と readReplyCounter の値が一致し，かつwriteRequestCounter と writeReplyCounter の値が一致するまで待つ．
3. バリア同期を実行

なお，メッセージパッシングを用いてダイレクトリモートデータアクセスをエミュレーションする場合には，ステップ2において，カウンタの検査と systemMessageHandler の呼びだしを交互に行なう．

5 評価

次の3つの観点から評価を行なった．

- ダイレクトリモートデータアクセスをメッセージパッシングライブラリを用いたエミュレーションで実現した方式で，割り込みハンドラによる実現に匹敵する性能が得られるか．
- 実行時にならないとデータ転送の組が決定できないようなプログラムに対してダイレクトリモートデータアクセスは全対全通信に比べて有利か．
- Fortran/AP におけるチューニングの効果はどのくらいあるか．（メッセージパッシングで直接プログラミングする場合に比較する．）

以下で，Speed up ratio とは，並列化する前のプログラムをAP1000の1プロセッサで実行させた処理時間を基準に何倍高速化されたかを示したものである．

5.1 ダイレクトリモートデータアクセスの実現法の違いによる性能差

図1にlist1で示した配列乗算のプログラムをAP1000を用いて実行させた結果を示す．図2にlist2で示したSPREAD MOVEを用いてチューニングを施した配列乗算のプログラムをAP1000を用いて実行させた結果を示す．send&rcv と示された線は，メッセージパッシングのライブラリを用いてダイレクトリモートデータアクセスを実現した場合の性能を示し，interrupt と示された線は，割り込みハンドラによりダイレクトリモートデータアクセスを実現した場合の性能を示している．

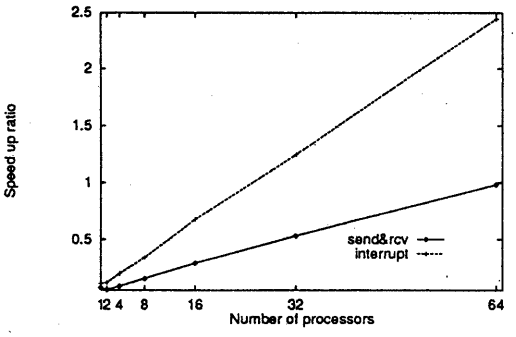

図 1: 配列乗算を用いたダイレクトリモートデータアクセスの性能比較

図1より，チューニングを施していない配列乗算のプログラムlist1については，全ての台数構成において割り込みハンドラで実現したものが，メッセージパッシングで実現したものに比べて2倍以上の速度が得られていることが分かる．どちらの実現についても速度の向上は小さく，64台のプロセッサを用いても割り込みハンドラ版で2.44倍の速度，メッセージパッシング版では0.98倍ともとの逐次プログラムより遅くなっている．

図2より，SPREAD MOVEを用いてチューニングを施した配列乗算のプログラムlist2については，逆に，メッセージパッシング版が割り込みハンドラ版を上回る性能を得ていることがわかる．どちらの実現についても，プロセッサ台数が16台までは，ほぼリニアな台数効果が得

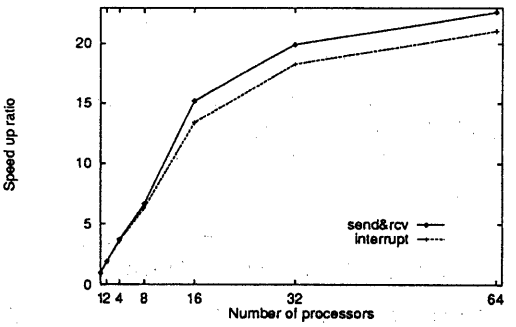

図 2: チューニングした配列乗算を用いたダイレクトリモートデータアクセスの性能比較

られている．

以上より，ダイレクトリモートアクセスをメッセージパッシングを用いてエミュレーションした場合は，割り込みハンドラで実現する場合に比べ，小さなデータによる通信が多数発生するようなプログラムにおいて性能が低いことが示された．しかしながら，どちらの実現法にしても小さなデータの通信が多数発生するプログラムでは高い性能が得られず，実用的な観点から通信の一括化は必須であるといえる．今回の実験では，通信の一括化によるチューニングが施されたプログラムに対しては，メッセージパッシングを用いたエミュレーション版が高い性能を出すことも示せた．

以下の節では，ダイレクトリモートアクセスの評価にメッセージパッシングライブラリを用いて実現した版で評価を行なう．

5.2 配列の添字に配列がくる場合の性能評価

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を検討する動機となった不規則な配列アクセスについての性能評価を，スケジュールされた send & receive によるコードと比較して行なう．

List 4 配列の添字に配列がくるプログラム

```
1    !XOCL SPREAD MOVE /(P)
2          DO 200 I = 1, N
3             AL(I)=B(IDX(I))
4     200  CONTINUE
5    !XOCL END SPREAD (X)
6    !XOCL MOVEWAIT (X)
```

評価には，list4のプログラムを用いる．配列Aと配列Bはプロセッサにブロック分割されて

おり，IDXは各プロセッサが全領域を持つ．実験の際に，IDXの各要素IDX(I)の値として，N-I+1 を初期設定した．スケジュールされたsend & receive によるコード生成としては，各プロセッサが担当する配列Bの要素を他の全てのプロセッサにブロードキャストし受信したプロセッサはそのうち必要なデータのみを配列ALに代入する方式を採用した．2種類の配列サイズについての実験結果を図3に示す．remote read 128Kbyte はダイレクトリモートデータアクセスにより，配列Bの大きさが128Kbyteの場合の転送を10回くりかえした実行時間(秒)を示している．broadcast 128Kbyte は，それのブロードキャスト版である．remote read 16Kbyte と broadcast 16Kbyte は，同様の実験を配列Bの大きさを16Kbyteにして行なったものである．

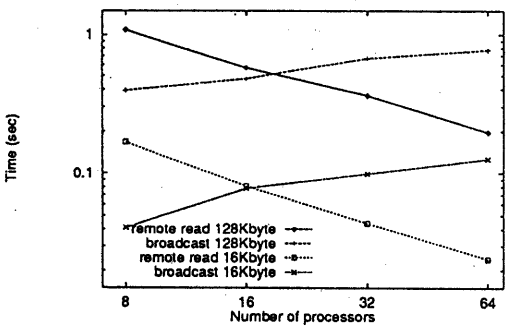

図 3: 配列の添字に配列がくるアクセス性能比較

2種類のサイズについてほぼ同じ傾向の結果が得られた．16台より少ない構成ではブロードキャストが有利であり，16台より多い構成ではダイレクトリモートアクセスが有利である．

ブロードキャストでは通信は送信側から受信側への一方向であるが，ダイレクトリモートアクセスのreadRemoteではread要求とそれに対する返答の双方向になるので，通信のオーバーヘッドが大きくなる．台数が少ない場合には，このオーバーヘッドによりダイレクトリモートアクセスのほうが遅くなる．一方，ブロードキャストは全プロセッサが行なわねばならないので，台数が多くなるにつれオーバーヘッドは大きくなる．

5.3 チューニングの効果

SPECベンチマークよりtomcatvをとりあげて並列化した．このプログラムは配列をプロセッサに分割する際にオーバーラップエリアを設けることで，OVERLAP FIX を用いてチューニングが可能なものである．

以下の3種類のプログラムについて実行時間の測定を行ない，並列化する前のもとのプログラムと比較した．

- 並列化可能なDOループに SPREAD DO のディレクティブを挿入したもの．
- さらに，OVERLAP FIX を用いて最適化したもの．
- メッセージパッシングプログラムで直接通信を記述したもの．

図4にAP1000を用いた測定結果を示す．

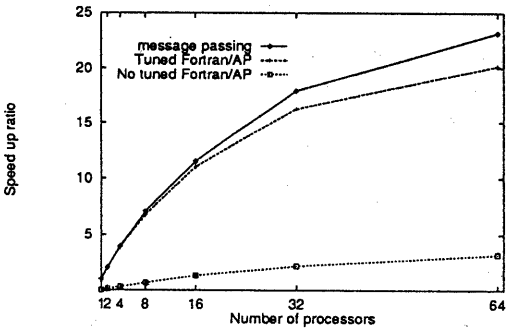

図 4: チューニングの効果

図4より，OVERLAP FIX による通信の一括化が効果があり，メッセージパッシングで直接記述した場合に比べ8台構成で95%，64台構成で87%の性能を得ている．

6 関連する研究

コンパイラにより分散メモリ型のメッセージパッシングマシン上でグローバルなアドレス空間をプログラマに提供する試みはOxygen[6], Kali[7], Fortran D[5]など多数なされている．これらのプロセッサ間通信法は，スケジュールされた send & receive により実現されており，配列の添字に配列がくるような不規則なデータアクセスに対して性能が出ないといった問題があった．この問題に対して，窪田[8]らは，特殊なケースに対して全対全通信を避ける手法として逆インデクス法と全検査法を提案している．一方，我々は，この問題に対する一般的な解決法として，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を提案した．

ダイレクトリモートデータアクセスに関連する研究としては，分散メモリをハードウェア

を用いて単一アドレス空間としてアクセスする方法として，StanfordのDASH[2]やNECのCenju[9]があり，分散メモリ間のデータ転送を行なう専用ハードウェアを用いる方式として富士通のVPP500[1]が実現されている．また，ソフトウェアでの実現としては，Eickenらによって active messages [3]としてCM5上での実験結果が報告されている．我々は，同等の機能をメッセージパッシングライブラリを用いて実現する方法を試みその実用性を示した．

7 まとめ

分散メモリ型並列計算機AP1000上でグローバルなアドレス空間をプログラマに提供するVPP Fortran の処理系(Fortran/AP)を実現した．

プロセッサ間にまたがるデータアクセスの実現法としてダイレクトリモートデータアクセスのインターフェイスを用いたコード生成を提案した．また，ダイレクトリモートデータアクセスの実現法として，割り込みハンドラによる方法と，メッセージパッシングライブラリを用いてエミュレーションする方法の2つを実験した．

実験結果より，以下の事を示した．

- チューニングされた VPP Fortran のプログラムに対して，メッセージパッシング系のライブラリを用いて実現したダイレクトリモートデータアクセスは，割り込みハンドラにより実現したものに劣らない性能が得られる．
- 配列の添字に配列があらわれる不規則なデータアクセスについて，プロセッサの構成台数が多くなるにつれてダイレクトリモートデータアクセスが有利になる．
- ベンチマークプログラム tomcatv について，通信一括化のためのディレクティブを用いて VPP Fortran で書くことにより，メッセージパッシングにより直接作成したプログラムの87%以上の性能が得られる．

参考文献

[1] K. Miura, M. Takamura, Y. Sakamoto, and S. Okada. "Overview of the Fujitsu VPP500 Supercomputer" in Proc. of COMPCON 93 pp.128-130, Feb. 1993.

[2] D. L. Lenoski, J. P. Laudon, K. Gharachorloo, A. Gupta, and J. L. Hennessy. "The directory-based cache coherence protocal for the DASH multiprocessor" in Proc. of the 17th Annual International Symposium on Computer Architecture pp.148-159, 1990

[3] T. V. Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. "Active Messages: a Mechanism for Integrated Communication and Computation" in Proc. of th 19th Annual International Symposium on Computer Architecture pp.256-266, 1992.

[4] High Performance Fortran Languages Specification Version 1.0. High Performance Fortran Forum, May 1993.

[5] S. Hiranandani, K. Kennedy, and C. Tseng. "Compiler optimizations for Fortran D on MIMD Distributed-Memory Machines" in Proc. of Supercomputing'91 pp.86-100, Nov. 1991.

[6] R. Rühl and M. Annaratone. "Parallelization of FORTRAN Code on Distributed-memory Parallel Processors" in Proc. of International Conference on SUPERCOMPUTING, pp 342-353, June 1990.

[7] C. Koelbel and P. Mehrotra. "Compiling global Name-Space Parallel Loops for Distributed Execution" in IEEE Transactions on Parallel and Distributed Systems, pp. 440-451, Oct. 1991.

[8] 窪田, 三吉, 大野, 森, 中島, 富田: "分散メモリ型並列計算機の自動並列化コンパイラ-Inspector/Executorアルゴリズムの高速化-", 並列処理シンポジウム JSPP '93, pp.47-54, May 1993

[9] 中田, 田辺, 梶原, 松下, 小野, 浅野, 小池: "並列回路シミュレーションマシンCenju", 情報処理学会30周年記念論文集, 31,5, pp593-601, May 1990.

[10] 石畑, 稲野, 堀江, 清水, 池坂: "高並列計算機AP1000のアーキテクチャ", 信学論 (D-I),J75-D-I, No.8, pp.637-645, Aug. 1992.