ハイパフォーマンス
コンピューティング 58－3
（1995．10．18）

並列言語 ADETRAN4 の VPP-500 での利用と評価

今村 俊幸　杉山 和徳　野木 達夫
京都大学 工学部 応用システム科学教室
〒606-01 京都市左京区

Tel: 075-753-5872　Fax: 075-761-2437
E-mail: {imamura,sugiyama,nogi}@kuamp.kyoto-u.ac.jp
WWW: http://fndsys.kuamp.kyoto-u.ac.jp/~imamura

科学技術計算に現れる様々な数値計算においてADI法の様な交互方向の作用素分割によって並列化がうまくなされるという視点から並列計算スキーム ADEPS と共にアーキテクチャー・ADENA ファミリーが野木により提唱されてきた。並列言語 ADETRAN4 は ADENA ファミリーの最新モデル ADENA4 上で利用するために考案された言語であり、ADETRAN をさらにユーザーフレンドリーにした上位言語である。本稿では並列言語 ADETRAN4 をベクトル並列計算機 VPP500 上でプリプロセッサの形で実装し、行列計算や NAS パラレルベンチマーク等の実際のプログラム開発を通しての実行効率やコーディングの容易さ等々を評価する。

Parallel Language ADETRAN4 — Its Utilization and Evaluation on VPP-500

Toshiyuki IMAMURA　Kazunori SUGIYAMA　Tatsuo NOGI
Division of Applied Systems Science, Kyoto University
Kyoto, 606-01, Japan

Tel: 075-753-5872　Fax: 075-761-2437
E-mail: {imamura,sugiyama,nogi}@kuamp.kyoto-u.ac.jp
WWW: http://fndsys.kuamp.kyoto-u.ac.jp/~imamura

At a point of view that in many scientific computations it is effective to use an alternating direction factorization such as ADI method, Nogi proposed the parallel computational scheme ADEPS and the architecture family ADENA. Parallel language ADETRAN4 is designed for ADENA4 which is the latest model of ADENA family and the upper version of ADETRAN. In this report we implement ADETRAN4 on vector-parallel supercomputer VPP-500 as a pre-processor. And through the actual programming, for matrix-multiplication and NAS parallel benchmark, we evaluate the efficiency of performance and easiness of coding.

1 はじめに

科学技術計算分野においてTFLOPSを標傍する次期スーパコンピュータとしてベクトルと並列の二つの技術を融合させたものが最有力であろう。野木は数値計算においてADI法の様な交互方向の作用素分割が並列化に重要な役割を果たすことに早くから注目し並列計算スキームADEPSとアーキテクチャADENAファミリーを提唱してきた[1, 2]。ADENAファミリーの最新モデルであるADENA4はベクトル・パラレル＋分散共有メモリ方式のアーキテクチャーである。並列言語ADETRAN4はこのADENA4上で利用するために考案された言語であり、実用機となったADENA2(ADENART256)上の言語ADETRAN [3, 4] をよりユーザフレンドリにした上位言語である。ADETRAN4の特徴はFortran90[5], HPF[6], Vienna-Fortran[7], Fortran-D[8]などと同様にFortranをベースとした言語であるが、並列拡張部分がコンパイラ指示子の形の実現ではなく完全な独自構文として提供されており、それらが並列プログラミングにおける重要な階層構造を形成している。そして並列計算スキームADEPSに従った処理では、通常データ転送や同期をコンパイラ側が解析し自動生成するのでユーザが意識しなくてもよい。またユーザーがデータを自由に編集できるような構文(分散メモリの場合データ転送を意味する)が用意されておりそれらはユーザー側でチューニングの手段として利用できる。

我々はADETRAN4を通した並列計算機の利用を模索しており様々なベクトルパラレルコンピュータ上でのインプリメントを計画している。今回はVPP-500(富士通)上でプリプロセッサの形でのインプリメントを行なった。

本稿ではまずADETRAN4について簡単な紹介をし、次にVPP-500上でのプリプロセッサの実装方法を示す。そして行列乗算計算やNASパラレルベンチマーク等実際のプログラミングと計算を通してADETRAN4での開発効率や実行効率を評価する。

2 並列言語ADETRAN 4

ADETRAN4は並列計算機ADENA4でのSPMD方式のプログラム言語であり、FORTRAN77に対してADENA4独自の並列処理形態を表現するデータ構造及び並列処理構文を追加したものである。

ADENA4の並列処理は**ADEPS**(**A**lternating **D**irection **E**xecution '**P**arallel over **S**egments')に基づくものであり、物理現象を表現する2次元又は3次元配列をある方向から見た1次元配列要素(セグメント)の集合と考えそれを処理の基本単位とし計算するものである。例えば2次元問題についてであれば、列↔行の2方向の処理をダイナミックに切替えながら処理を進行する様式である。また、異なる位置にあるセグメント同志の計算ができないという制約を設けることで処理中のグローバルアクセスの弊害をなくし、処理表現が一貫しまた簡潔なものにできる。

次にADETRAN4の構文について説明する。ADETRAN4はホスト－スレーブ方式におけるスレーブ部分のプログラムを記述するものである。ADETRAN4の構文は次の3階層に分かれており、システム全体のグローバル制御(G), 並列実行制御指示(P), 各プロセッサ単位のローカル構文(L)のG-P-Lが各構文の先頭文字につき区別される(Lの一部は除く)。

[G] GDO,GIF,GGOTO,GCALL,IFALL/IFANY文
[P] PDO,PCAST文
[L] FORTRAN77における実行文(入出力を除く)

これらの階層化された構文はそれ以下の階層を含むことで全体が形成される。またこれらの階層に対応したサブプログラムがあり、図2.1の関係をなす。

ADETRAN4の持つデータ構造は2もしくは3次元のセグメント配列(またはそれらを拡張した配列)とセグメントベクトルの2つである。前者は全データを分散して持ちシステム全体で一つの配列と見るもの、後者は個々のプロセッサ単位で重複して持つ配列である。これらデータはシステム構成によって実装は異なるが、基本的に1次元のセグメントに分割したものがプロセッサに静的に割り当てられる。

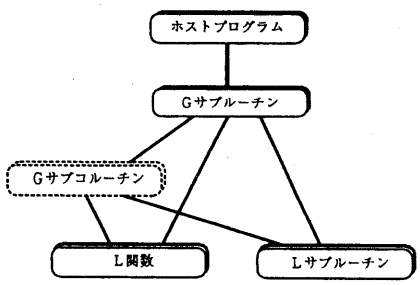

図2.1 サブプログラムの階層構造

```
pdo j=1,n
  do i=2,n-1
    v(i,/j/)=u(i-1,/j/)+u(i+1,/j/)
  enddo
pend
pdo i=1,n
  do j=2,n-1
    v(/i/,j)=u(i,j)+a*(u(i,j-1)\
              -2*u(i,j)+u(i,j+1)+v(i,j))
  enddo
pend
```

図 2.2　2 次元拡散スキーム

分割されたデータは軸方向毎のアクセスに対しローカルとなるようコンパイラ側でデータ転送を行なうなど整合性が保たれる。ユーザーは軸方向毎に配列を確保する必要も軸方向を変えることで生じるデータ転送を記述する必要もない。

並列計算の実行は **PDO~PEND** ブロック内で行なわれる。個々の PDO ブロック内は一つの**方向属性**を持ち、その方向と同じ属性を持つ配列のみ計算がなされる。この制限は物理的イメージと対応が付けられよう。図 2.2 は典型的な 2 次元拡散スキームを ADETRAN 4 で記述したものである。このプログラムの様にユーザーは方向属性にあるセグメントの 1 インデックスを自由に変化させながらプログラミングをする。そして残りのインデックスは並列の分割かまたはベクトルの実行ループとされる。この処理様式が ADEPS であり、したがってそれを表現する ADETRAN4 はベクトル機上での開発に向いているといえる。

文献 [1] では様々な問題レベルでの ADEPS での表現を取り上げているが、交互方向に変数を見ながら DO ループを回すだけでは配列の特定場所をアクセスするのに不自由な場合が起きる。それを補うために **PCAST 文**というデータ編集構文が提供されている。PCAST 文は分割されたデータの一部をローカルな配列 (セグメントベクトル) に編集しなおすもので、例えば行列の転置やリストベクトルを用いたアクセスが異なるプロセッサに及ぶなどのものに対して使われる。これらのアクセスをグローバル空間やメッセージパッシングを使って処理する言語もあるが、処理の形態として遅延の蓄積によってもっともパフォーマンスが悪いものである。ADETRAN4/ADEPS では必要なデータはユーザの責任で確保し、連続処理を効果的に得ようという立場をとっている。この点は他の言語と若干意識の違いをユーザに求める部分である。

以上が ADETRAN4 と ADEPS の概要である。

3　VPP-500 での実装

今回 ADETRAN4 コンパイラをネイティブコンパイラとしてではなく ADETRAN4→VPP-Fortran のプリプロセッサとして実装した。プリプロセッサのパス構成は次の 4 段である。

1. ADETRAN4 パーサー
2. ブロック内方向属性解析
3. データ転送最適化
4. コード生成器

以下プリプロセッサの変換規則について説明する。

3.1　データ分散と並列実行

ADETRAN4 における方向属性は VPP-Fortran では図 3.1 のように方向属性そしてグローバル・ローカルのタグを変数名に付加し識別・管理する。ここで配列の添字がローテションしているのは各方向毎にベクトル化率や実行速度のばらつきをなくすためであり、分割軸の位置はベクトル実行効率の最も良い第 3 インデックスにし、第 1 インデックスにベクトル実行の添字部分を割り振った。例えば図 3.1 の始めの PDO ブロックにおける i がベクトル実行部分で j が分割指定部分となる。残りのインデックスは VPP コンパイラでは逐次実行となる。またプロセッサへの割り当ては固定で BAND とした。

```
pdo  i=1,n,j=1,n
  A(/i,j/,1)=A(/i,j/,n)
pend
   :
pdo j=1,n,k=1,n
  do i=1,n
   A(i,/j,k/)=A(i,/j,k/)+1
  enddo
pend
```

↓

```
!XOCL SPREAD NOBARRIER DO /P3
  do j=1,n
  do i=1,n
    $d3 A(i,i,j)=$d3_A(i,n,j)
  enddo
  enddo
!XOCL END SPREAD NOBARRIER
   :
!XOCL SPREAD NOBARRIER DO /P1
  do k=1,n
  do j=1,n
   do i=1,n
    $d1 A(j,i,k)=$d1_A(j,i,k)+1
   enddo
  enddo
  enddo
!XOCL END SPREAD NOBARRIER
```

図 3.1　データ分散と並列実行部分の実装

3.2 データ転送

異なる方向属性のデータを VPP-Fortran 側で軸を転置した変数として持ち整合性を保たせるので、図 3.2 のような変数の定義・参照があった時にデータ転送の必要が生じる。コンパイラはこのような DEF-USE で方向属性の異なるものを検出し適当な場所にデータ転送を挿入する。我々はこの転送を PASS と呼ぶ。PASS の検出は個々の PDO ブロック内の USE,DEF を調べ、方向属性を付加し拡張した LV 問題と RD 問題の積集合上の path でデータ転送の回数を最小にするループ深度の最も浅い位置でかつソースに近い場所に PASS を挿入する。VPP では転送命令と終了確認命令の間に通常の計算を挟むことができ通信と計算を並行してできる。そこで上記の PASS の挿入点から実際に USE されるまでの最も遠い点に修了確認の同期命令 (SYNC) を挟む (図 3.3)。このことにより通信を計算の裏側に隠すことができる。実装は SPREAD MOVE と MOVEWAIT で行なう。

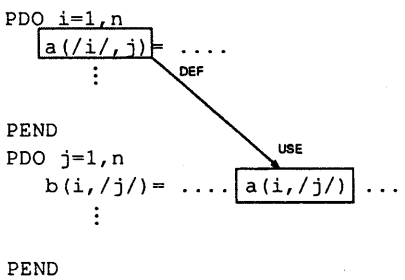

図 3.2 異なる方向属性間の USE-DEF の例

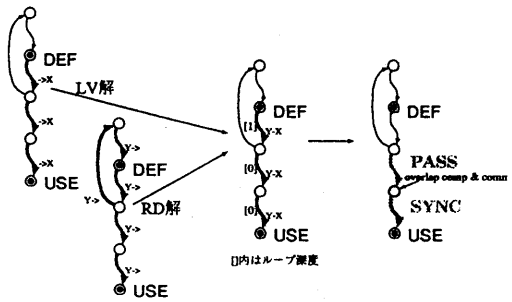

図 3.3 PASS,SYNC 挿入アルゴリズム

3.3 サブプログラム

ホストプログラムからのみ呼び出される G サブルーチンは PARALLEL REGION~END PARALLEL を含む並列化手続きとして変換し、G サブコルーチンはスプレッド手続きとした。今回これら手続き間の変数受渡しは common ブロックを用いたもののみで行ないプロセッサは固定とした。また L サブルーチン・L 関数は PURE な手続きとして変換し、変数の受渡しは引数とセグメントベクトルのみを含む common ブロックの二方法を可能とした。ホスト部分は通常の VP-Fortran の記述になるが変数名の管理の都合上、変数の受渡し部分の仕様が定まらず未実装となってしまった。入出力などが必要となった時に応じて変換されたプログラムを手で書き換え対応した。

4 評価

今節では VPP-500/15 上で実際に ADETRAN4 を使ったプログラミングを行ない、その開発効率と実行効率について述べる。実行においては都合上プロセッサ数は最大 10 台 (ピーク性能 16.0GFLOPS) で行なっている。

4.1 行列乗算

List 1a は行列の乗算サブルーチンの FORTRAN77 版である。これと等価なサブルーチンを ADETRAN4 で記述したものが List 1b である。List 1a では b(k,j) の形の参照があるがこれはローカルなアクセスでは実現されないので ADETRAN4 では pcast 構文を用いてローカルな配列に編集し直している。pcast 文と中間変数を除くと両者はほぼ同じである。

List 1b を注意して見ると pcast 構文において添字の順序を入れ換えている。これは乗算実行時にローカル配列へアクセスが最も効率的になる順序であり HPF には見られないベクトル化における中間変数の

List 1a. 行列乗算 (FORTRAN77)

```
1   subroutine matmul
2   parameter ( n=1000 )
3   real a(n,n),b(n,n),c(n,n)
4   common /m/a,b,c
5   do i=1,n
6     do j=1,n
7       c(i,j) = 0.0
8     enddo
9     do k=1,n
10      do j=1,n
11        c(i,j)=c(i,j)+a(i,k)*b(k,j)
12      enddo
13    enddo
14  enddo
15  return
16  end
```

= List 1b. 行列乗算 (ADETRAN4) =

```
 1  gsubroutine matmul
 2  parameter ( n=1000 )
 3  region ( n,n )
 4  real a(n,n),b(n,n),c(n,n),t(n)(n)
 5  common /m/a,b,c
 6  pcast i=1,n, j=1,n
 7    t(j)(i) = b(i,j)
 8  pend
 9  pdo i=1,n
10    do j=1,n
11      c(/i/,j) = 0.0
12    enddo
13    do k=1,n
14      do j=1,n
15        c(/i/,j)=c(/i/,j)+a(/i/,k)*t(j)(k)
16      enddo
17    enddo
18  pend
19  return
20  end
```

チューニングの自由度をユーザに解放しているといえる。実際添字を入れ換えたList 1bでは14.8GFLOPSとピーク性能(16.0GFLOPS)の92.5%を出したのに対し添字を入れ換えないものでは3.17GFLOPSとピークの1/5程度しか出すことができない。

4.2 NASパラレルベンチマーク：SP

NASパラレルベンチマーク[12](以下NPB)からCFD-application:SP(Scalar Pentadiagonal solver)を選んでADETRAN4でプログラミングおよび計算を行なった。プログラムコードはNAS Ames Research centerのサンプルをから始める形で進め、ベクトル化に適した書換えを行なっていった。我々はすでにADENART256でNPBを行なった経験があったが[15]、VPP-500でのベクトル化という問題がありADENART256で開発したコードは参考程度にしかならなかった。しかしながらSPのアルゴリズム自体が交互方向の作用素分解であったためにコード化はそれほど難しいことではなかった。実際3000行のサンプルコードのADETRAN4化をほぼ3日で終らせることができた、デバッグも合わせるとトータルで4日を要し、その後のチューニング等も合わせると約1週間の開発期間であった。List 2に開発コードの一部(サンプルコードのrhsに相当)を示す。

実行結果については全体で400回ループうちの1回分のみの計算時間を算出し、それを適時チューニングを施していきながら400回ループで40.6秒(2.85GFLOPS)を記録した。Fujitsu AmericaによるNPBの結果[13, 14]を見る限り、この数字はその2倍強の時間でありコンパイラ側・ユーザ側双方でのチューニングの余地を残す結果となった。

= List 2. NPB-SP(ADETRAN4) =

```
 1  gsubcoroutine grhs
 2  include "gappsp.incl"
 3  real flux(*,*,*)(5)
 4  pdo j = 1, ny, k = 1, nz
 5    do l = 1, 5
 6      do i = 1, nx
 7        rsd(i,j,k)(l) = - frct(i,j,k)(l)
 8      enddo
 9    enddo
10  pend
11c *** xi-direction flux differences
12  pdo j = 2, ny1, k = 2, nz1
13    do i = 1, nx
14      flux(i,j,k)(1) = u(i,j,k)(2)
15      u21 = u(i,j,k)(2) / u(i,j,k)(1)
16      q = 0.50 * (  u(i,j,k)(2)**2 \
17                  + u(i,j,k)(3)**2 \
18                  + u(i,j,k)(4)**2 ) \
19                  / u(i,j,k)(1)
20      flux(i,j,k)(2) = u(i,j,k)(2) * u21 \
21                     + c2*(u(i,j,k)(5) - q)
22      flux(i,j,k)(3) = u(i,j,k)(3) * u21
23      flux(i,j,k)(4) = u(i,j,k)(4) * u21
24      flux(i,j,k)(5) = ( c1*u(i,j,k)(5) \
25                     - c2*q ) * u21
26    enddo
27  pend
28  pdo j = 2, ny1, k = 2, nz1
29    do l = 1, 5
30     do i = 2, nx1
31       rsd(i,j,k)(l) =  rsd(i,j,k)(l) \
32        - tx2*(flux(i+1,j,k)(l)-flux(i-1,j,k)(l))
33     enddo
34    enddo
      .....
```

5 その他開発環境について

5.1 デバッグ

ADETRAN4プリプロセッサはEWS(Sparc Station 10)上で開発をし、実行もEWS上で行なわれる。そこで通常のプログラム編集作業なども全てEWS上で行い、実行の段階でVPP-500システムにftpなどで転送をし、計算を行なう方法をとっている。ADETRAN4からVPP-Fortranに変換したプログラムの利点としてコンパイラ指示子を完全なコメントとしてみなしても逐次実行において殆んど等価なFORTRAN77プログラムとなる点にある[9]。この利点を活かしVPPのフロントエンドUXPのみならず標準的なEWS上のFORTRAN77コンパイラや豊富なデバッグツールを用いることができる。EWS上で開発・デバッグできる点はユーザにとって大きなメリットである。

5.2 チューニング

VPPの性能を十分に発揮するには個々のプロセッサでのベクトル化が十分になされなければいけない。

ユーザーはプリプロセッサで変換されたVPP-Fortranプログラムを通じてVPPコンパイラにベクトル化の促進がなされるようにベクトル計算機における一般的なチューニングを行なう。今回の研究では次の方法が有効であった。

1. 計算をまとめたりDOループをタイトにしベクトルレジスタの効率をあげる。
2. 多次元配列の宣言範囲を変えバンク競合を減らす。
3. ベクトル化促進・抑制指示子(VPP-Fortran用)を積極的に入れる。

上記のチューニングを行なうことで10%から50%の速度向上が見られる。特にNPBのSPにおいて1の項目は適応範囲が多く効果は大きかった。

6 今後の課題

今回、サブプログラム呼び出しにおいて引数をcommonブロックによる引渡しを行なう実装にしたが、この場合commonブロックで共有する方向をあらかじめ決めておく必要がありサブプログラム内で冗長なデータ転送が生じる。実際NPBのSPでメインループからcallされる4つのサブプログラムを一つのサブプログラムにまとめたところデータ転送回数が2/3となり実行時間は35.5秒と10%の性能向上([14]の約2倍)が見られた。このことからinterprocedual analysisを行ない共有commonブロックの方向属性についての最適化を施していく必要があると考える。またデータ転送を変数の全ての領域で行なっているが予め更新および参照される領域が分かっている場合にはその部分のみの転送をすることで通信の最適化が図れると考える。またホストプログラムとの変数受渡しを実装し入出力をスムーズにする必要がある。

さらに実行速度面から、プリプロセッサ側でループの解析をより強化しベクトル化を促進するようなVPP-Fortranコードを生成するとともに、VPPコンパイラの生成するコードの性質にも踏み込みプログラム変換のロスを最小にする必要があると考える。

7 まとめ

並列言語ADETRAN 4をVPP-500上でインプリメントし、簡潔でかつ一貫したプログラミング環境が得られた。そしてその有効性はプログラム開発の容易さに現れることを示した。またベンダーのベンチマークには及ばないものの1/2程度の実行速度が容易に得られることが示された。今後はVPP-500のみならず他のシステムでのインプリメントを行なっていきたい。

最後に、本研究にあたり京都大学大型計算機センターのVPP-500/15を利用した。

参考文献

[1] 野木 達夫:「並列計算モデルPB対ADEPS」, 応用数理 Vol.3, No.3, pp.29-46, 1993.

[2] T.Nogi: "Promising Data Parallel Environment — ADEPS, ADETRAN & ADENA —," Proc. pAs'95 in Aizu, 1995.

[3] T.Nogi: "Parallel Language ADETRAN," Mem. Fac. Eng. Kyoto Univ. Vol.51, No.4, pp.235-290, 1989.

[4] H.Kadota, etc.: "Parallel computer ADENART —its architecture and application." Proc. Inter. Conf. on Supercomputing, pp.1-8, 1991.

[5] J.C.Adams, etc.: "FORTRAN90 Handbook Complete ANSI/ISO Reference," McGrowHill, 1993.

[6] High Performance Fortran Forum :"High Performance Fortran Language Specification," version 1.1, Nov. 1994.

[7] H.Zima, etc.: "Vienna Fortran – a Language Specification version 1.1," ACPC/TR 92-4. Mar. 1992.

[8] G.Fox, etc.: "Fortran D Language Specification," CRDC-TR 90079, Dec. 1990.

[9] 岩下 英俊, 他: "VPP Fortran: 分散メモリ型並列計算機言語," JSPP '94, 1994.

[10] 富士通マニュアル,「UXP/M VPP-FORTRAN77 EX/VPP 使用手引書」,1994.

[11] H.Zima :"Supercompilers for Parallel and Vector Computers," Addison-Wesley, 1991.

[12] D.Bailey, etc.: "The NAS Parallel Benchmarks," RNR-94-007, Mar. 1994.

[13] D.Bailey, etc.: "NAS Paeallel Bechmark Results 10-94," NAS-94-001, Oct. 1994.

[14] S.Saini, etc.: "NAS Paeallel Bechmark Results 3-95," NAS-95-011, Apr. 1995.

[15] T.Imamura, etc.: "NAS Parallel Benchmark Results on ADENART," Proc. Inter. Conf. PCFD'94.