［大規模科学計算システム］高速化推進研究活動報告第６号より転載

ベクトルコンピュータにおける高速化

小林広明[1]　江川隆輔[1]　小松一彦[1]　岡部公起[1]　大泉健治[2]　小野　敏[2]　山下　毅[2]
佐々木大輔[2]　森谷友映[2]　齋藤敦子[2]　撫佐昭裕[3]　松岡浩司[3]　渡部修[3]　曽我　隆[4]　山口健太[4]

[1] スーパーコンピューティング研究部
[2] 情報部情報基盤課
[3] 日本電気株式会社
[4] NEC ソリューションイノベータ株式会社

本章では，本センターにおいてこれまで培われてきた SX-9 のベクトル性能を向上するための手法や事例を紹介する．これらの手法や事例は今後導入されるベクトルスーパーコンピュータ SX-ACE においても有用である．

1.　ベクトルコンピュータの特徴

近年，高性能計算機（HPC）システムを活用したシミュレーションの応用範囲は益々広がっており，同時にシミュレーションの規模も拡大し続けている．シミュレーションを行う研究者からの要求は計算規模を拡大すると同時に，計算結果を得るまでの時間は短縮したいというものである．このような研究者の要求を満たすため HPC システムの性能も年々向上し続けている．

HPC システムに採用されるプロセッサあるいはコア(以下コア)には大きく分類して，スカラ型とベクトル型がある．スカラ型コアは命令レベル並列性に基づきデータを一つずつ実行するのに対して，ベクトル型コアはデータレベルの並列性に着目して複数の演算器(パイプライン)を用いて一度に複数の処理を同時に実行する．図 1-1 にスカラ命令とベクトル命令の動作を示す．この図は A(I)=B(I)+C(I)と D(I)=E(I)+F(I)の 2 つの計算式を 100 回繰り返す DO ループの処理を表している．スカラ命令では DO 変数が 1 の時の A(1)と D(1)の計算結果を求め，次に DO 変数が 2 の時の A(2)と D(2)の計算結果を求めるという処理を，DO 変数が 100 になるまで繰り返す．それに対してベクトル命令は A(I)=B(I)+C(I)の計算を DO 変数 1 から 100 まで一度に行い，次に D(I)=E(I)+F(I)の計算を DO 変数 1 から 100 までを一度に行うことにより実行時間の短縮を図る．ベクトル型コアを搭載する HPC システムは，このベクトル命令の特性を活かすため，高いメモリ帯域を具備してメモリからコアへの大量のデータ供給を可能にしている．

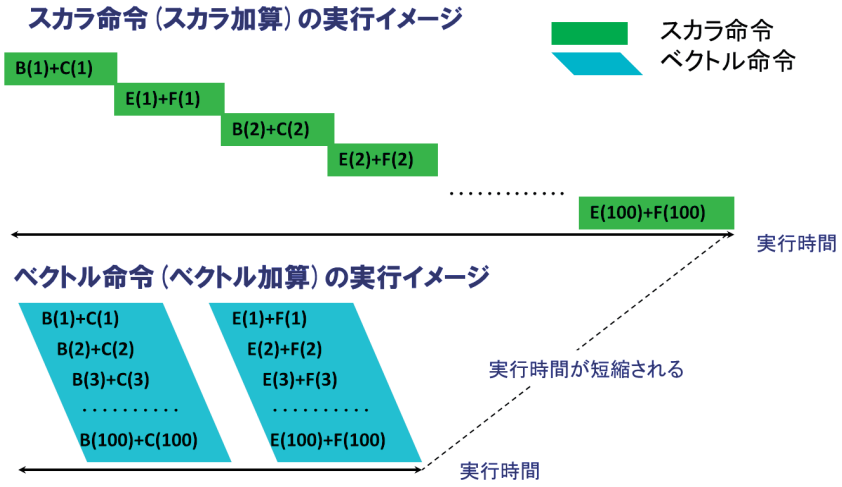

図 1-1　スカラ命令とベクトル命令の様子

表 1-1 に近年の HPC システムが搭載しているプロセッサの性能と内包するコア数，プロセッサ当たりのメモリバンド幅を示す．SX-9 以外のプロセッサでは複数の演算コアを搭載するマルチコア構成により演算性能の向上を図っているため，コア当たりのメモリバンド幅は小さくなる．しかしスカラプロセッサではメモリのバンド幅を最大限に使うためには複数のコアを使用する必要があるが，SX-ACE では 1 つコアで最大のメモリバンド幅を使うことが可能となっている．

このようにベクトルコンピュータは，高い理論演算性能とメモリ帯域を実装することで，実アプリケーションの実行において高い実効性能の達成を可能にしている．

表 1-1 HPC システムのプロセッサ性能と B/F 値

HPC システム名	プロセッサ			メモリバンド幅 (GB/s)
	名称	理論性能(GFlop/s)	コア数	
SX-9	—	102.4	1	256
LX406Re-2	Intel Xeon E5-269v2	230.4	12	59.7
SR16000	POWER7	245.1	8	128
FX10	SPARC64 IXfx	236.5	16	85.3
「京」	SPARC64 VIIIfx	128.0	8	64
SX-ACE	—	256.0	4	256

2.　高速化の手法

SX システムの超高速性を十分に引き出すために重要なことは，プログラムの中でベクトル命令によって処理される部分の比率，すなわちベクトル化率と，生成されるベクトル命令の効率を可能な限り高めることである．そのためには，プログラムの最適化状況の把握・分析を行い，主要な性能指標に基づき，高速化を行う必要がある．ここでは，そのためのツール類及び，それらの性能指標と各指標に沿った高速化の取り組みかたについて述べる．

2.1 性能解析ツール

(1)　編集リスト

編集リストはベクトル化および自動並列化に関する情報をソースプログラムの左側に表示したリストであり，どのループがベクトル化されたかを確認することができる．編集リストはコンパイラオプション「-R5」を指定することにより"入力ファイル名.L"という名前で出力される．

```
!編集リスト
FILE NAME:t5.f
PROGRAM NAME: sub
FORMAT LIST
 LINE    LOOP     FORTRAN STATEMENT
    1:               subroutine sub(a,b,c,d,z,ix)
    2:               real,dimension(100)::a,b,c,d,x,y,z
    3:               integer ix(100)
    4: V------>        do I = 1,100
    5: |       I          call sub2(x,a,b,I)
    6: |                  y(I) = c(I) + d(I)
    7: |       S          z(ix(I)) = z(ix(I)) + x(I) + y(I)
    8: V------         enddo
    9:               end
```

図 2-1　編集リストの例

主なループ情報，スカラ情報，手続インライン情報など編集リストの出力イメージを次に示す．

①　ループ全体がベクトル化される場合

ベクトル化されたループに"V"が表示される．

```
V------>   do I=1,100
|            a(I)=b(I)+c(I)
V------    enddo
```

図 2-2　ベクトル化された DO ループの編集リスト

②　ベクトル化されない場合

ベクトル化されないループには"+"が表示される．

```
+------>   do I=1,100
|            print *,a(I)
+------    enddo
```

図 2-3　ベクトル化されない DO ループの編集リスト

③ 部分ベクトル化の場合

ベクトル化不可の処理がある行には“S”が表示される.

※部分ベクトルとは，ループにベクトル化を阻害する部分が含まれている場合，その前後で自動的にループを分割し，ベクトル化可能な部分だけをベクトル化する拡張機能である.

```
V------>   do I =1,100
|            a(I)=b(I)+c(I)
|      S     print *,a(I)
V------      enddo
```

図 2-4 部分ベクトル化の編集リスト

④ 配列式をベクトル化した場合(1)

ループの先頭と最後の行が同じである場合，ループの構造は“=”で表示される.

```
             real a(90),b(90),c
 V=======    a(1:90)=b(1:90)+c
```

図 2-5 ベクトル化された配列式の編集リスト

⑤ 配列式をベクトル化した場合(2)

ループ融合している場合は，その範囲について“V”で表示される.

※ループ融合とは，繰返し数が等しい DO ループまたは配列式が複数個，連続して存在している場合一つのループに融合することである. ただし，各ループで使われているデータの定義・参照関係に，融合によって矛盾が生じる場合は行わない.

```
            real a(90),b(90),c
           integer d(90),e(90)
V------>    a(1:90)=b(1:90)+c
|           d(1:90)=int(a(1:90))
V------     e(1:90)=d(1:90)+1
```

図 2-6 ループ融合とベクトル化された配列式の編集リスト

⑥ 手続呼出しがインライン展開された場合

インライン展開された手続がある行には“I”が表示される.

```
         I  call sub2(x,a,b,c,I)
```

図 2-7 インライン展開の編集リスト

⑦ 多重ループが一重化された場合

一重化されたループの外側ループに“W”，内側ループに“＊”が表示される.

```
W------>   do J =1,100
|*---->      do I =1,100
||             a(I,J)=b(I,J)+c(I,J)
|*----       enddo
W------     enddo
```

図 2-8 多重 DO ループの一重化の編集リスト

⑧ ループの入れ換えが行われた場合

入れ換えた結果ベクトル化されるループに“X”が表示され，ベクトル化されなくなるループには“+”が表示される.

```
X------>    do J =1,1000
|+------>     do I =1,10
||              a(I,J)=b(I,J)+c(I,J)
|+------       enddo
X------      enddo
```

図 2-9 DO ループの入れ換えの編集リスト

⑨ ADB にデータがバッファリングされる場合

ADB を使用したベクトルロード/ストアがある行に文字“A”が表示される.

```
+------>    do J =1,100
|V------>    do I =1,100
||      A      a(I,J)=a(I,J+1)+b(I,J)
|V------      enddo
+------      enddo
```

図 2-10 ADB の使用の編集リスト

⑩ ベクトル化と並列化される場合

ベクトル化と並列化が行われたループに“Y”が表示される.

```
Y------>      do i=1, 10000
|               c(i) = c(i) + a(i) * b(i)
Y------       end do
```

図 2-11 ベクトル化と並列化された DO ループの編集リスト

⑪ 並列化される場合

並列化が行われたループに“P”が表示される.

```
P------>          do j = 1, 100
|V------>          do i = 1, 100
||                   d(i,j) = 0.0d0
|V------            end do
P------           end do
```

図 2-12 並列化された DO ループの編集リスト

⑫ 複数の情報がある場合

一行に対して複数の情報がある場合, “M”が表示される.

※配列 a はベクトル化され, 配列 b はループ長が短いためループが展開されるので, この一行には複数の情報がある.

```
M======           a(1:10000)=0.0d0 ; b(1:3)=0.0d0
```

図 2-13 複数の情報がある場合の編集リスト

(2) 簡易性能解析機能(FTRACE)

簡易性能解析情報はコンパイラオプションに「-ftrace」を指定することで, 手続(サブルーチンや関数)ごとの性能解析情報を採取することができ, プログラム実行後に解析情報が標準出力ファイルに出力される. 特に, チューニング対象とする手続を選択する際に活用すると良い. 図 2-14 に FTRACE 機能による解析リストの例を示す. FTRACE では実効性能と平均ベクトル長などの情報を手続(サブルーチンや関数)単位で取得することができる.

① PROC.NAME	② FREQUENCY	③ EXCLUSIVE TIME[sec](%)	④ AVER.TIME [msec]	⑤ MOPS	⑥ MFLOPS	⑦ V.OP RATIO	⑧ AVER. V.LEN	⑨ VECTOR TIME	⑩ I-CACHE MISS	⑪ O-CACHE MISS	⑫ BANK CONFLICT CPU PORT	⑫ BANK CONFLICT NETWORK
subc	100	7.594(42.2)	75.936	79426.5	52676.9	99.48	256.0	7.594	0.0000	0.0000	0.032	0.000
subb	100	5.227(29.1)	52.272	77121.3	38261.9	99.22	256.0	5.227	0.0000	0.0000	0.000	0.000
suba	100	5.173(28.7)	51.728	58600.1	1933.29	98.97	256.0	5.173	0.0000	0.0000	0.000	0.000
test	1	0.000(0.0)	0.518	772.0	0.0	0.00	0.0	0.000	0.0000	0.0000	0.000	0.000
total	301	17.994(100.0)	59.780	7276.83	38902.9	99.28	256.0	17.993	0.0001	0.0001	0.032	0.000

図 2-14 解析リストの出力例

表示される各項目の意味は以下のとおりである.

①PROC.NAME :手続名
②FREQUENCY :手続の呼び出し回数
③EXCLUSIVE TIME :手続の実行に要した占有の CPU 時間(秒)と, 手続全体の実行に要した CPU 時間に対する比率
④AVER.TIME :手続の 1 回の実行に要した平均 CPU 時間(ミリ秒)
⑤MOPS :1 秒間に実行された演算数を 100 万単位で示した値
⑥MFLOPS :1 秒間に実行された浮動小数点演算数を 100 万単位で示した値
⑦V.OP RATIO :ベクトル演算率
⑧AVER V.LEN :平均ベクトル長
⑨VECTOR TIME :ベクトル命令実行時間(秒)
⑩I-CACHE MISS :命令キャッシュミスにより発生した合計時間(秒)
⑪O-CACHE MISS :オペランドキャッシュミスにより発生した合計時間(秒)
⑫BANK CONFLICT :バンクコンフリクト時間(秒)
CPU PORT :CPU ポート競合時間(秒)
NETWORK :メモリネットワーク競合時間(秒)

2.2. ベクトル化率の向上

(1) ベクトル化率

ベクトル型スーパーコンピュータではプログラム中のベクトル処理可能な部分を高速に実行している. 図 2-15 に同一のプログラムをスカラ処理する場合とベクトル処理する場合の実行時間に関する概念図を示す.

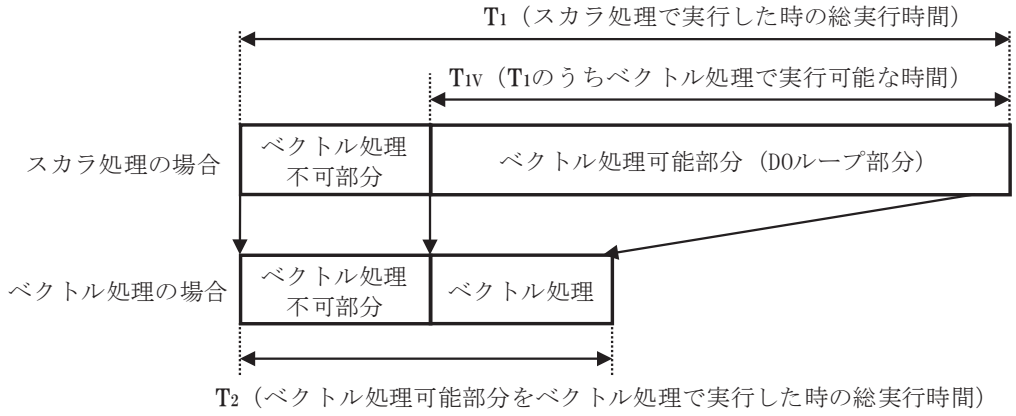

図 2-15　ベクトル処理による実行時間短縮のイメージ

ここで，あるプログラムをスカラ処理で実行する場合の総実行時間を T_1，そのプログラムでベクトル処理が可能な部分の実行時間を T_{1V} とすると，ベクトル化率 α は以下の式で定義される．

$$\text{ベクトル化率}\quad \alpha = \frac{T_{1V}}{T_1}$$

また，そのプログラムをベクトル処理する場合の性能向上比 P は，スカラ処理性能とベクトル処理性能の比を β とすると，

$$\text{性能向上比}\quad P = \frac{1}{(1-\alpha)+\frac{\alpha}{\beta}}$$

と表される．

この式から，ベクトル化率と性能向上比の関係は図 2-16 のようになる（アムダールの法則）．この図からベクトル化率80%程度では大きな性能向上は見られず，ベクトル化率90%を超えたあたりから急激に性能が向上していることがわかる．ベクトル型スーパーコンピュータで高い実効性能を得るためにはベクトル化率を 100%にできる限り近付ける必要がある．

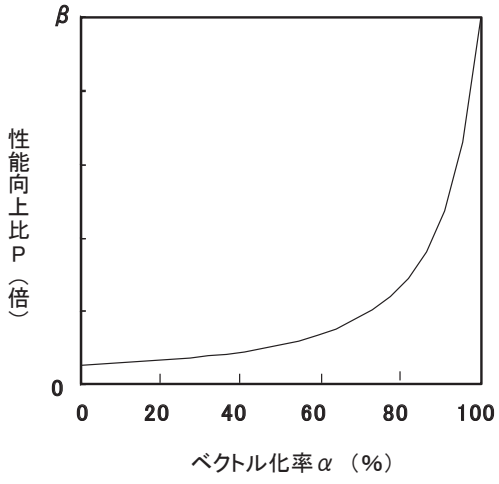

図 2-16　ベクトル化率と性能向上比（アムダールの法則）

ベクトル化率を求めるため，プログラムをスカラ実行する場合とベクトル実行する場合のそれぞれの実行時間が必要である．しかし，これらの実行時間は同時に得られないのでベクトル化率の算出は困難である．そのため SX-9 ではベクトル化率の代わりにベクトル演算率をベクトル化の指標としている．FTRACE などの性能解析機能に出力される情報はベクトル演算率であり，ベクトル演算率は，プログラムで処理される全演算要素数に対するベクトル演算命令で処理される演算要素数の割合で算出し，ほぼベクトル化率とみなせる指標であるため，後述のベクトル化率はベクトル演算率のこととする．

(2) ベクトル化を阻害する要因

ベクトル化を阻害する要因を以下に示す．

① 依存関係の有無が判断できない
※前の繰り返しで定義した値を参照する場合や間接参照

② I/O 処理(OPEN/CLOSE/READ/WRITE 文)

③ ユーザ関数(SUBROUTINE, FUNCTION)の呼び出し

④ ループの繰り返し数がループ本体の実行前に決定しない

⑤ ループの入口および出口が一つではない

⑥ ベクトル化対象外の精度を使用している
※2 バイト整数型，4 倍精度実数型，4 倍精度複素数型，1 バイト論理型，文字型,構造型はベクトル化対象外の精度

2.3 平均ベクトル長の拡大

(1) 平均ベクトル長

ベクトル処理を効率的に行う上で重要な指標にベクトル長がある．ベクトル長はベクトル化対象の DO ループの繰り返し回数のことであり，効率良くベクトル処理を行うためには，できるだけループ長を長くする必要がある．

一般に命令を発行してから結果が返ってくるまでに遅延時間が存在する．この遅延時間を立ち上がり時間と呼ぶ．図 2-17 にベクトル処理における立ち上がり時間および演算時間の概念図を示す．網掛け部分は演算器が稼働する時間を示しており，演算を行うレジスタのデータを読み込み，演算結果をレジスタに格納するまでの時間となる．図 2-18 にベクトル長と立ち上がり時間の関係を示す．立ち上がり時間はベクトル長が異なる場合でも一定である．よって総演算量が等しい場合，短ベクトル長処理を繰り返すよりも長ベクトル長処理を一括して行う方が実行時間を短くすることができる．このように高速化を図るためには，DO ループのベクトル長を十分に確保し，演算時間に対する立ち上がり時間の占める割合を低くすることが重要である．図 2-19 にベクトル長と立ち上がり時間の関係を示す．ベクトル処理の立ち上り時間が発生するため，ベクトル長が交差ループ長より短い(4 以下)場合はベクトル化を行わない方が良い場合がある．交差ベクトル長とは，ベクトル化した場合とベクトル化しない場合とで実行時間が等しくなるループ長のことである．

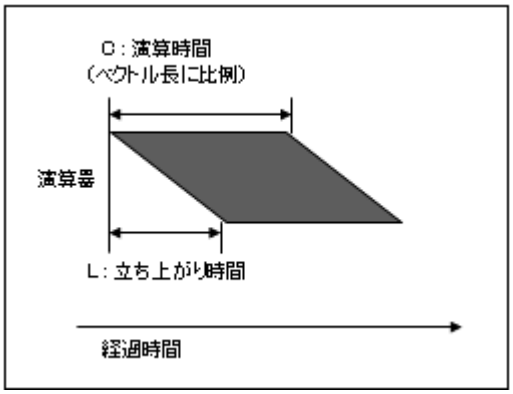

図 2-17　立ち上がり時間

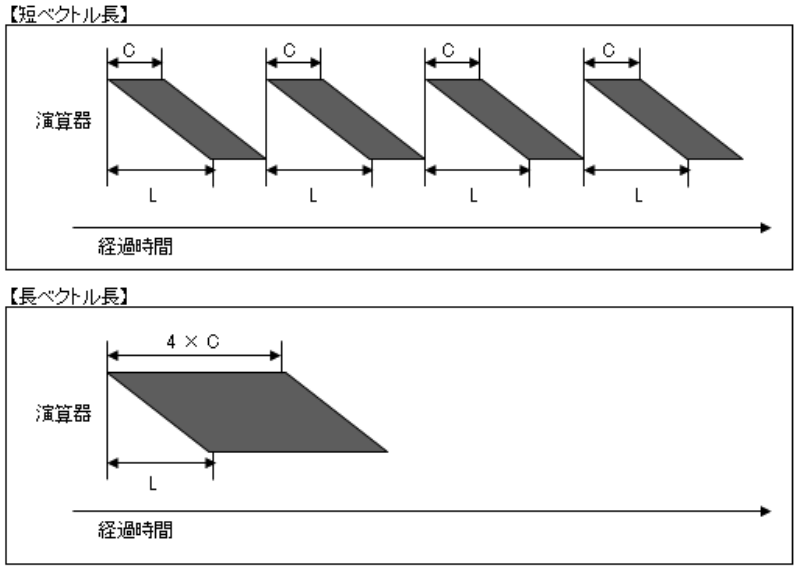

図 2-18　ベクトル長と立ち上がり時間

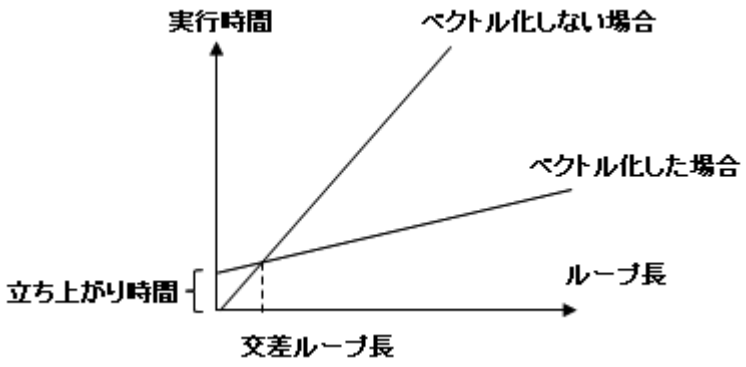

図 2-19　ベクトル長と立ち上がり時間の関係

2.4 メモリアクセスの効率化

(1) メモリ競合

SX-9 ではマルチメモリバンクシステムを採用しており，メモリを 32 のバンクグループに分割している．各バンクグループには 2 要素(16 バイト)ずつ配列データが格納される．

図 2-20 は連続アクセスとなる DO ループの例，図 2-21 は格納される配列データ(二次元の配列サイズが(128,64)のとき)の概念図を示しており，CPU からメモリにアクセスする際，CPU 内の全ポートを使用して効率的にメモリアクセスを行う．

```
 !連続アクセス
do j=1,ny
do i=1,nx
  :
  Rs(i,j)=Rs(i,j)+Cf(i,j)
end do
end do
```

図 2-20 連続アクセスの DO ループの例

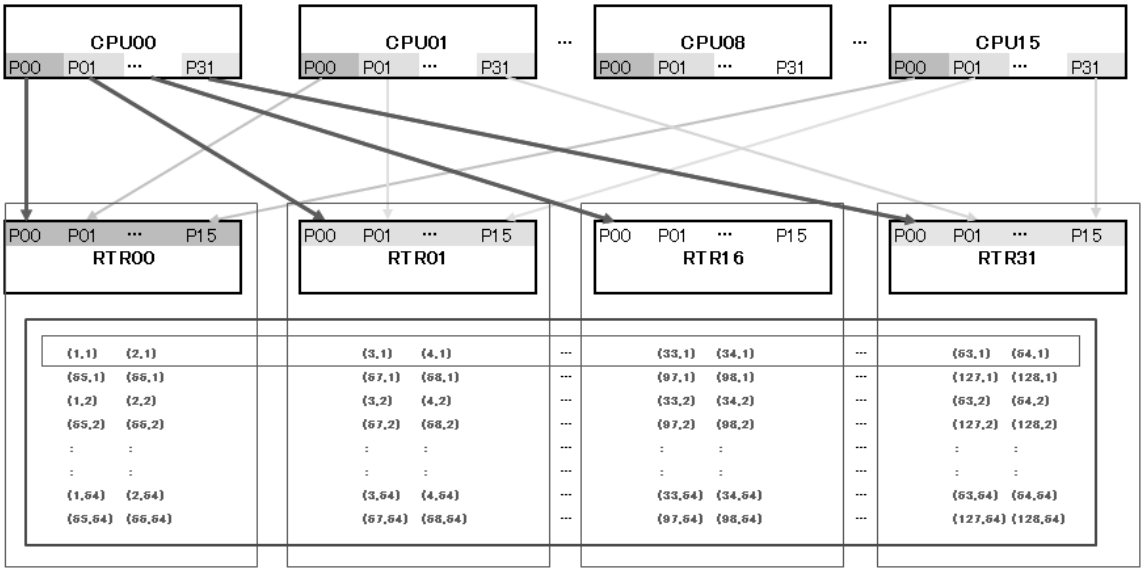

図 2-21 メモリバンクに格納される配列データの概念図(連続アクセス)

一方，図 2-22 はストライドアクセスとなる DO ループの例，図 2-23 は格納される配列データ(二次元の配列サイズが(128,64)のとき)の概念図を示しており，CPU からメモリをアクセスする際，アクセスするバンクグループが限定され CPU 内の特定のポート(00, 16)だけを使用してメモリアクセスを行うことになる．このように CPU 内における同一のポートにロード・ストアが集中した時に発生する CPU ポート競合や，同一のメモリバンクへのアクセスなどで発生するメモリネットワーク競合はメモリアクセスが遅延するため高速化の妨げとなる．

```
!ストライドアクセス
do  j=1,ny
do  i=1,nx,32
   :
  Rs(i,j)=Rs(i,j)+Cf(i,j)
end do
end do
```

図 2-22 ストライドアクセスの DO ループの例

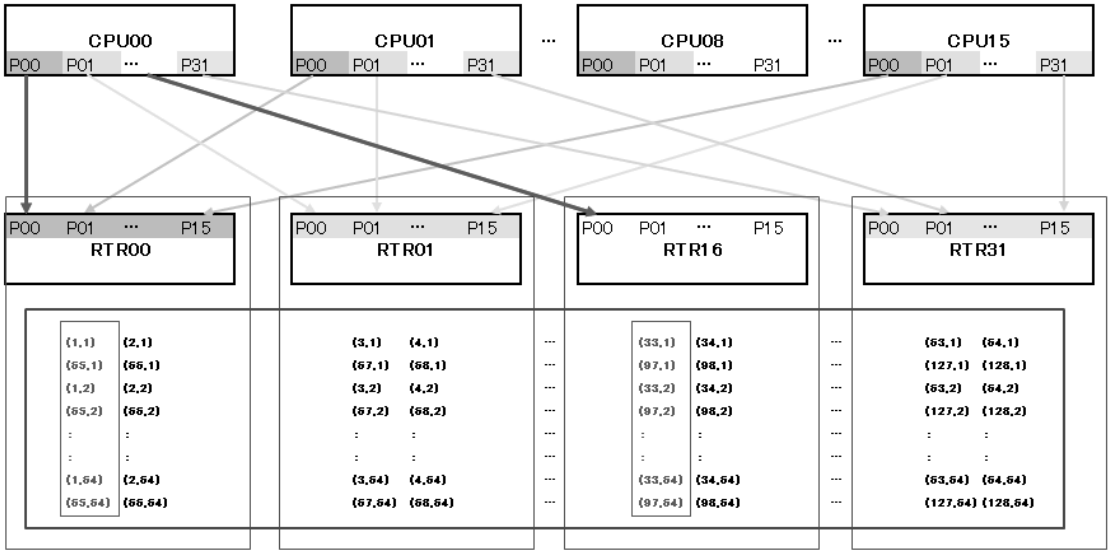

図 2-23 メモリバンクに格納される配列データのイメージ(ストライドアクセス)

以降にメモリアクセス性能を改善するための指針として，メモリアクセスパターンの改善と ADB の活用について述べる．

(2) メモリアクセスパターンの改善

図 2-24 にメモリへアクセスパターンの種類を示す．アクセスパターンの種類としては，連続アクセス，ストライドアクセス，間接アクセスの 3 種類に分けることができる．

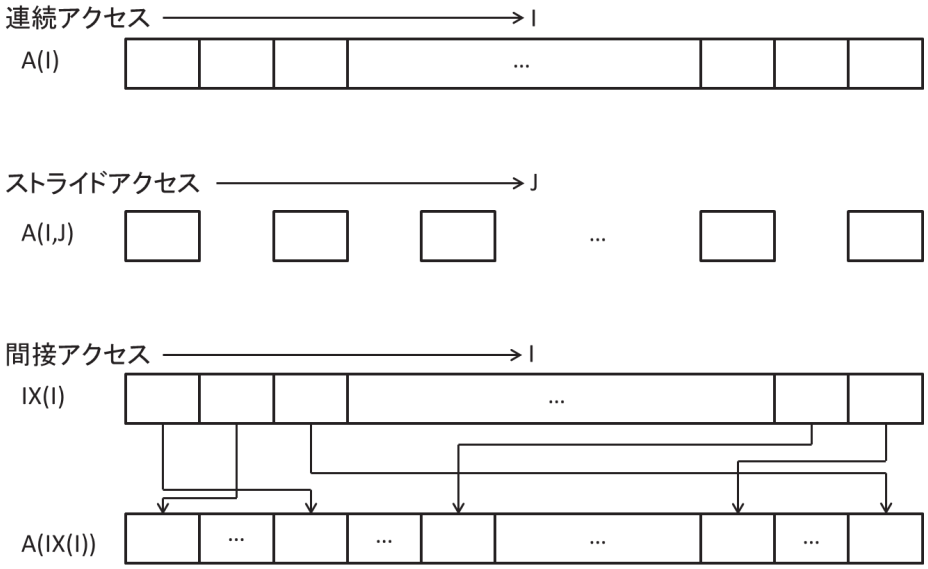

図 2-24 メモリのアクセスパターンの種類

図 2-21 に示すように，連続アクセスの場合が最も効率的なメモリアクセスを行うことができる．ストライドアクセスの場合には，図 2-23 に示すように，使用できるバンク数が限られてくる．この場合，ループ入れ換えなどにより，なるべく最初の次元でメモリアクセスを行うようにする．その際，ループインデックスが繰り返しごとに 1 ずつ増加あるいは減少するようにできでば連続アクセスとなり，より効率的なメモリアクセスが可能となる．間接アクセスの場合，間接アクセスとなる配列(図 2-24 の間接アクセスにおける配列 A)が何度も参照されるならば，最初に，この配列のデータを作業用の配列にコピーしておき，以降のループでは作業用の配列をアクセスするようにすることにより，連続アクセスにすることができる．

(3) ADB の活用

SX-9 では CPU-主記憶装置間に ADB(Assinable Data Buffer)と呼ばれるベクトルデータを選択的にバッファする機能を有している．この ADB を活用することによって，メモリアクセス性能を改善できる．

コンパイラは，再利用性があると判断した配列を自動的に ADB にバッファリングするが，コンパイラが判断できなかった配列については，ON_ADB 指示行を用いることによって，明示的に該当配列を ADB にバッファリングすることを指示することができる．例えば，間接アクセスにおいて，間接アクセスとなる配列(図 2-24 の間接アクセスにおける配列 A)に対し ON_ADB 指示行を用いて ADB にバッファリングしておくことにより，この配列のロードを高速に行えるようになる．

なお，ADB の容量は限られているため，ADB にバッファリングするデータのサイズに注意し，再利用性のあるデータを ADB にバッファリングするように指定することが重要である．

3.　ベクトル化率向上の事例

3.1. ループ内の WRITE 文の括り出し

(1)　チューニング方針

SX-9 では，DO ループの一部にベクトル化対象外の文が存在すると，ベクトル化されない．もしくは，部分ベクトル化となり，完全にはベクトル化されない．　図 3-1 にチューニング前のコードを示す．このコードでは，DO ループ内にエラー検出時に実行される WRITE 文が含まれている．入出力文はベクトル化対象外の処理であるため，DO ループ全体がベクトル化されていない．図 3-2 のチューニング前の FTRACE 情報を確認すると，ベクトル化率が 0.0%であり，全くベクトル化されていないことが分かる．そこで，ベクトル化阻害要因である WRITE 文を DO ループの外に移動させることにより，DO ループをベクトル化可能にする．

```
!チューニング前
12: +------>          DO J=2, JF
13: |+----->             DO I=2, IF
 :
18: ||                     IF (HZ(I,J) .GE. -30.0) THEN
19: ||                        ZZ = Z(I,J,1) - RX*(M(I,J,1)-M(I-1,J,1))
20: ||                &                     - RY*(N(I,J,1)-N(I,J-1,1))
21: ||                        IF (ABS(ZZ) .LT. GX) ZZ = 0.0
22: ||                        DD = ZZ + HZ(I,J)
23: ||                        IF (DD .LT. GX) DD = 0.0
24: ||                        DZ(I,J,2) = DD
25: ||                        Z(I,J,2)  = DD - HZ(I,J)
28: ||                        IF(ABS(Z(I,J,2)).GT.100.0) THEN
29: ||                        NC=1
30: ||                        WRITE(*,'(A24,3I6)')'Over flow Z at (K,I,J) :', KK,I,J
31: ||                        WRITE(*,*)'Within Region :', NREG
32: ||                        WRITE(*,*)'Computation is unstable.'
34: ||                        END IF
36: ||                     END IF
37: |+-----             END DO
 38: +------          END DO
```

図 3-1　WRITE 文括り出し前のコード

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
40	6.863(3.1)	171.564	558.3	195.4	0.00	0.0	0.000	0.000	2.239	0.000	0.000	【チューニング前】

図 3-2　WRITE 文括り出し前の FTRACE 情報

(2)　チューニング内容

図 3-3 にチューニング後のコードを示す．チューニング前の DO ループではエラー検出時に WRITE 文によるエラー情報の出力を行っているが，本修正においては，DO ループ内ではエラー検出時にフラグ検出用配列にフラグをセットするようにした．そして，WRITE 文によるエラー情報の出力処理は DO ループ外に移動させ，出力が必要な場合のみ処理を行うようにした．これにより，DO ループ内にはベクトル化対象外の文がなくなるため，ベクトル化が可能となる．

```
!チューニング後
! フラグの初期化
23:                    IFLG1 = IF * JF + 1
24: W*=====            IDX = IFLG1
 :
! メイン処理
!   エラーの検出のみ実施して，エラー出力処理をループの外に出す
27: +------>           DO J=2, JF
28: |V----->              DO I=2, IF
 :
33: ||                       IF (HZ(I,J) .GE. -30.0) THEN
34: ||                          ZZ = Z(I,J,1) - RX*(M(I,J,1)-M(I-1,J,1))
35: ||                 &                      - RY*(N(I,J,1)-N(I,J-1,1))
36: ||                          IF (ABS(ZZ) .LT. GX) ZZ = 0.0
37: ||                          DD = ZZ + HZ(I,J)
38: ||
39: ||                          IF (DD .LT. GX) DD = 0.0
40: ||                          DZ(I,J,2) = DD
41: ||                          Z(I,J,2)  = DD - HZ(I,J)
 :
45: ||                          IF(ABS(Z(I,J,2)).GT.100.0) THEN
53: ||                             IDX(I,J) = I + (J - 1) * IF
54: ||                          END IF
56: ||                       END IF
57: |V-----              END DO
58: +------            END DO
 :
! エラー出力処理
61: +------>           DO J=2, JF
62: |V----->              DO I=2, IF
63: ||                       IFLG1 = MIN(IDX(I,J), IFLG1)
64: |V-----              ENDDO
65: +------            ENDDO
66:                    IF ( IFLG1 .LT. IF * JF + 1 ) THEN
67:                       INDX_J = (IFLG1 - 1) / IF + 1
68:                       INDX_I = IFLG1 - (INDX_J - 1) * IF
69:                       NC=1
70:
71:                       WRITE(*,'(A24,3I6)')'Over flow Z at (K,I,J) :',
73:                       WRITE(*,*)'Within Region :', NREG
74:                       WRITE(*,*)'Computation is unstable.'
75:                    ENDIF
```

図 3-3　WRITE 文括り出し後のコード

(3)　性能分析

図 3-4 にチューニング前後の性能値を示す.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CPU PORT	CONFLICT NETWORK	PROC.NAME
40	6.863(3.1)	171.564	558.3	195.4	**0.00**	0.0	0.000	0.000	2.239	0.000	0.000	【チューニング前】
40	0.079(0.0)	1.966	63568.9	18871.6	**99.54**	238.9	0.078	0.000	0.000	0.004	0.012	【チューニング後】

図 3-4　WRITE 文括り出し前後 FTRACE 情報

DO ループ全体がベクトル化されたことにより，ベクトル化率が 0.0%から 99.5%に向上し，実行時間は 6.86 秒から 0.08 秒に短縮することができた.

3.2. 部分ベクトル化の回避

(1) チューニング方針

図 3-5 にチューニング前のコードを示す. 39 行目の変数 vmax は前の繰り返しで定義された値を参照しているためベクトル化の阻害要因となるが, コンパイラは最大値を求める演算をマクロ演算に置き換えてベクトル化を行おうとする. 最大値もしくは最小値を求める演算がある場合, ベクトルパイプライン毎にベクトル演算を行い, 最後に各演算結果のマスクを取るマクロ演算を適用してベクトル化を行う. しかし, 39 行目で求める vmax の値を 40 行目で参照する必要があるため, ベクトル化を阻害する依存関係となり, 39 行目と 40 行目をスカラ処理する部分ベクトル化となる. 図 3-6 に示すチューニング前の FTRACE 情報を確認するとベクトル化率が 67.5%で十分にベクトル化がされていないことが分かる. ベクトル化を阻害する依存関係を排除し, マクロ演算を適用してループ全体をベクトル化する.

```
!チューニング前
23: +------>          do iy = 1,ny
24: |V----->            do ix = 1,nx
 :
37: ||                     absv =max(absvxi,absvxj,absvyi,absvyj)
38: ||                     cf   = sqrt(cs2(ix,iy)+ca2(ix,iy))
39: ||       S             vmax = max(vmax,absv+cf)
40: ||       S             dtmin = min(dtmin,dlmin/vmax)
41: |V-----              enddo
42: +------            enddo
```

図 3-5 部分ベクトル化回避前のコード

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
821	26.020(7.2)	31.693	1664.6	496.5	**67.53**	255.9	2.017	0.001	1.503	0.003	1.113	**【チューニング前】**

図 3-6 部分ベクトル化回避前の FTRACE 情報

(2) チューニング内容

チューニング前では vmax の最大値判定と更新, dtmin の最小値判定と更新をループ回数分行っている. dtmin の判定式は不変値 dlmin を vmax で除算しているため, dtmin は vmax が更新されたときのみ dtmin が更新される. そのため, DO ループで vmax の最大値を求め, DO ループ終了後に一回 dtmin の判定を行うことで同じ結果が得られる. 図 3-7 にチューニング後のコードを示す. 前述したとおり, MAX 関数で求める vmax を MIN 関数で参照することが依存関係であるため, MIN 関数の演算を DO ループの外に移動することで依存関係が解消され, vmax を求める処理はマクロ演算が適用されてベクトル化が行われる.

```
!チューニング後
23: +------>          do iy = 1,ny
24: |V----->            do ix = 1,nx
 :
37: ||      A              absv = max(absvxi,absvxj,absvyi,absvyj)
38: ||                     cf   = sqrt(cs2(ix,iy)+ca2(ix,iy))
39: ||      A              vmax = max(vmax,absv+cf))
40: |V-----              enddo
41: +------            enddo
42:                    dtmin = min(dtmin,dlmin/vmax)
```

図 3-7 部分ベクトル化回避後のコード

(3) 性能分析

図 3-8 にチューニング前後の性能値を示す.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CPU PORT	CONFLICT NETWORK	PROC.NAME
821	26.020(7.2)	31.693	1664.6	496.5	**67.53**	255.9	2.017	0.001	1.503	0.003	1.113	【チューニング前】
821	2.273(0.8)	2.769	14592.6	4215.6	**99.19**	255.9	1.971	0.000	0.124	0.019	1.029	【チューニング後】

図 3-8 部分ベクトル化回避前後の FTRACE 情報

部分ベクトルが解消しベクトル化率が 67.5%から 99.2%に改善したことで実行時間は 26.0 秒から 2.3 秒に短縮することができた.

4. 平均ベクトル長の拡大の事例

4.1. ループに関する情報の追加による最適化の促進

(1) チューニング方針

多重ループの最内にあるループが自動ベクトル化の対象となるため, 最内ループのベクトル長が短い場合, 十分な性能を発揮できない. 図 4-1 のチューニング前のコードではループの中に配列式や配列関数を使用しており, その部分が最内ループとなりベクトル化の対象となる. 図 4-2 の FTRACE 情報では平均ベクトル長が 67.0 であり十分なベクトル長でないことが分かる. ここでは, コンパイラがループ長を判断してベクトル長を伸ばす方法で最適化を行う. ループの演算処理量が多いなどの問題からこのような最適化が行われない場合もあるが, ループに関する情報をコンパイラに明示することでベクトル長を伸ばす最適化が行われる場合がある. このような高速化の事例を以下に示す.

```
!チューニング前
 5:            real*8:: y(Nch), xp(Ndim,Nch,Np)
 6:            real*8:: loglhTP(Np), dy(Nch)
 :
15: V------>  do ip=1,Np
16: |V===== A   dy = y - xp(1,:,ip)  &
17: |                          - xp(Mu+1,:,ip) - xp(Mu+Ms,:,ip)
18: |
19: |V===== A   loglhTP(ip) = - 0.5d0 * ( &
20: |                          dble(Nch) * log( 2.0d0 * pi )            &
21: |                        + log( product( xp(Ndim,:,ip) ))           &
22: |                        + sum( dy * dy / xp(Ndim,:,ip) )           &
23: |                          )
24: V------   enddo
```

図 4-1 ループに関する情報を追加する前のコード

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
60	73.899(73.1)	1231.647	2404.9	576.8	89.76	**67.0**	63.853	0.000	0.000	0.001	41.961	【チューニング前】

図 4-2 ループに関する情報を追加する前の FTRACE 情報

(2) チューニング内容

ベクトル化対象となる配列式と配列関数のループ長(変数 Nch)は 3, 外側 ip の DO ループの長さは 1,280,000 であり, この情報を追加することでコンパイラはより効率の良い最適化を実施する. 図 4-3 にチューニング後コードを示す. PARAMETER 文を追加することでコンパイラは最内 DO ループのループ長が 3 であることを認識するため, ループの展開を行い, 外側のループでベクトル化を行う. ただし, このチューニング内容は変数 Nch, Np の値が不変であることを前提としている場合のみ使用することが出来るので注意が必要となる.

```
!チューニング後
 6:            parameter(Nch=3,Np=1280000)
 7:            real*8:: y(Nch), xp(Ndim,Nch,Np)
 8:            real*8:: loglhTP(Np), dy(Nch)

15: V------> do ip=1,Np
16: |*=====    dy = y - xp(1,:,ip)
17: |               - xp(Mu+1,:,ip) - xp(Mu+Ms,:,ip)
18: |
19: |*=====    loglhTP(ip) = - 0.5d0 * ( &
20: |                         dble(Nch) * log( 2.0d0 * pi )            &
21: |                       + log( product( xp(Ndim,:,ip) ))           &
22: |                       + sum( dy * dy / xp(Ndim,:,ip) )           &
23: |                         )
24: V------  enddo
```

図 4-3 ループに関する情報を追加後のコード

(3) 性能分析

図 4-4 にチューニング前後の性能値を示す.

```
 FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS    MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE   BANK CONFLICT   PROC.NAME
           TIME[sec](  %)     [msec]                      RATIO V.LEN      TIME    MISS    MISS CPU PORT  NETWORK

       60   73.899( 73.1)  1231.647   2404.9    576.8 89.76  67.0     63.853   0.000   0.000    0.001    41.961 【チューニング前】
       60    0.289(  1.0)     4.813  23763.3  12232.7 99.60 256.0      0.267   0.000   0.000    0.017     0.111 【チューニング後】
```

図 4-4 ループに関する情報を追加する前後の FTRACE 情報

外側のループをベクトル化対象とすることで平均ベクトル長が 67.0 から 256.0 となり, 十分な性能を発揮することで, 実行時間が 73.9 秒から 0.3 秒に高速化された.

4.2. 部分配列を DO ループに書き替え, ループ交換を実施

(1) チューニング方針

チューニング前コードおよびチューニング前の FTRACE 情報は前項と同じ図 4-1 と図 4-2 である. 前項では, 変数 Nch, Np の値が不変であるという条件のもと PARAMETER 文を追加することでコンパイラの最適化を促進させた. ここでは変数 Nch, Np の値が不定の場合のチューニング方法を示す.

(2) チューニング内容

変数 Nch, Np の値が不定の場合のチューニング方法を示すが, ここで Nch の値が小さく, Np の値が十分に大きいことが分かっているものとする. 図 4-5 にチューニング後コードを示す. 配列式と配列関数を DO ループの形に変形し, ループ分割を行っている. このとき作業配列を新たに用意し使用する. また, Nch＜Np であるため, Np の DO ループがベクトル化の対象となるように最内にループの入れ換えを行っている.

```
!チューニング後
 5:            real*8:: y(Nch), xp(Ndim,Nch,Np)
 6:            real*8:: logIhTP(Np), dy(Nch)
 8:            real*8:: wrk_dy,wrk
 9:            real*8:: wrk_sum(Np),wrk_prd(Np)

24: V------>   wrk_prd=1.d0
25: V------    wrk_sum=0.d0
26: +------>   do j=1,Nch
27: |V----->   do ip=1,Np
28: ||      A    wrk_dy=y(j)-xp(1,j,ip)                    &
29: ||                 -xp(Mu+1,j,ip)-xp(Mu+Ms,j,ip)
30: ||      A    wrk_prd(ip)=wrk_prd(ip)                   &
31: ||                    * xp(Ndim,j,ip)
32: ||      A    wrk_sum(ip)=wrk_sum(ip)                   &
33: ||                  +(wrk_dy**2/xp(Ndim,j,ip))
34: |V-----    end do
35: +------    end do
36:            wrk=dble(Nch)*log(2.0d0*pi)
37: V------>   do ip=1,Np
38: |       A    logIhTP(ip)=-0.5d0*(wrk                  &
39: |                +log(wrk_prd(ip))+wrk_sum(ip))
40: V------    enddo
```

図 4-5 部分配列を書き換え後のコード

(3) 性能分析

図 4-6 にチューニング前後の性能値を示す.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
60	**73.899**(73.1)	1231.647	2404.9	576.8	89.76	**67.0**	63.853	0.000	0.000	0.001	41.961	【チューニング前】
60	**0.314**(1.1)	5.232	25804.5	11742.5	99.54	**256.0**	0.287	0.000	0.000	0.026	0.101	【チューニング後】

図 4-6 部分配列を書き換え前後の FTRACE 情報

外側のループをベクトル化対象とすることで平均ベクトル長が 67.0 から 256.0 となり, 実行時間が 73.9 秒から 0.3 秒に短縮された.

5. メモリ競合の回避の事例

5.1. ADB によるメモリアクセス低減

(1) チューニング方針

0.(3)で述べているように, ADB を利用することにより, より効率的なメモリアクセスを行うことができる. その事例として, 図 5-1 を挙げる.

図 5-1 において最内側の DO ループがベクトル化されているが, 配列 b はこのベクトル化されたループに入る度にメモリからのロードを行うため, メモリ競合が発生する. 図 5-2 の FTRACE 情報が示すように, チューニング前のコードでは, 実行時間 26.6 秒に対してバンクコンフリクト時間が 18.1 秒を占めている. この配列 b に着目してみると, ベクトル化されている DO ループの外側の DO ループ(DO 変数 l のループ)による繰り返しによって配列の参照位置が変わることはなく, 最内 DO ループにおいては, 同じ添え字 j のデータを毎回メモリからロードすることが分かる. 従って, 配列 b は最内 DO ループにおいて再利用性があることが分かる.

通常, コンパイラはループに対する最適化処理の中で配列の再利用性解析を行う. しかし, 本事例のように, ループからの飛び出しがあるような場合にはコンパイラはループの最適化処理を行わない. そのため, 配列の再利用性解析も行われず, 再利用性がある配列に対してもコンパイラは自動でADBに載せるための処理を行わない. このような場合, コンパイラ指示行により明示的に配列を ADB に載せることを指示し, ADB 経由でのメモリアクセスを行うことにより, メモリに直接アクセスする頻度を削減し, バンクコンフリクト時間を短縮する.

```
!チューニング前
335: ||+----->              do l=1,lmax
336: |||V--->                 do i=is,imax(j)+1
337: ||||                        if(a(l).lt.b(i,j)) exit
338: |||V---                   end do
339: |||                       is    =i
340: |||                       iwk(l)=i
341: ||+-----               end do
```

図 5-1 ADB によるメモリアクセス低減前のコード

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
126	26.574(100.0)	210.906	1166.8	342.1	88.72	114.4	22.488	0.000	0.000	0.012	**18.072**	【チューニング前】

図 5-2 ADB によるメモリアクセス低減前の FTRACE 情報

(2) チューニング内容

ON_ADB 指示行を用いて，配列 b を ADB に載せることを指示する．チューニング後のコードを図 5-3 に示す．これにより 1 回目のロード命令はメモリから行われるが，2 回目以降のロード命令は ADB から行われることになる．この結果，メモリへのアクセスを削減し，バンクコンフリクトによる実行時間の増加を解消する．

```
!チューニング後
335: ||+----->              do l=1,lmax
336: |||          !cdir on_adb(b)
337: |||V--->                 do i=is,imax(j)+1
338: ||||    A                    if(a(l).lt.b(i,j)) exit
339: |||V---                   end do
340: |||                       is    =i
341: |||                       iwk(l)=i
342: ||+-----                 end do
```

図 5-3 ADB によるメモリアクセス低減後のコード

(3) 性能分析

図 5-4 にチューニング前後の性能値を示す．

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CPU PORT	CONFLICT NETWORK	PROC.NAME
126	26.574(100.0)	210.906	1166.8	342.1	88.72	114.4	22.488	0.000	0.000	0.012	**18.072**	【チューニング前】
126	9.547(99.9)	75.771	3247.7	952.3	88.72	114.4	5.455	0.000	0.000	0.012	**1.036**	【チューニング後】

図 5-4 ADB によるメモリアクセス低減前後の FTRACE 情報

メモリへのアクセスが減少することにより，バンクコンフリクト時間が短縮され，実行時間が 26.6 秒から 9.5 秒に短縮することができた．

5.2. ループ融合によるベクトルロード・ストアの削減

(1) チューニング方針

通常，コンパイラによる最適化では，同じループ構造の DO ループが連続していると自動でループ融合を行う．ループ融合を行うことで同一の配列データに対するメモリアクセス(ロード，ストア)回数を減らすことができる．図 5-5 にチューニング前のコードを示す．チューニング前コードでは同じループ構造の DO ループが連続しているが OpenMP 指示行を含んでいるためループ融合が行われない．コンパイラは PARALLEL リージョン単位で最適化を行うため，ループ融合が行われず連続する複数の DO ループで同じ配列データのメモリアクセスが発生し，配列 sm3dh はメモリロード 2 回，メモリストア 2 回の命令が発生している．図 5-6 の FTRACE 情報を確認すると実行時間 13.1 秒に対してバンクコンフリクト時間が 7.4 秒を占めているため，ループ融合によりメモリアクセス回数を減らしバンクコンフリクト時間が短縮される．

```
!チューニング前
7414:              !$OMP PARALLEL SHARED (sm3dh,sn3dh,sl3dh,sr3dh,se3dh
  :
7419:              !$OMP DO
7420: P------>           do iz=1,lz
7421: |+----->           do iy=1,ly
7422: ||V---->           do ix=3,lx-2
  :
7437: |||     A             sm3dh(ix,iy,iz)=sm3d(ix,iy,iz)
7438: |||               &             +sdltxinv*( du(ix-1,iy,iz)-du(ix,iy,iz) )
  :
7443: ||V----            enddo
7444: |+-----            enddo
7445: P------            enddo
7446:              !$OMP END DO NOWAIT
7447:              !$OMP END PARALLEL
  :
7454:                    sdltthalfgx=0.5d0*adltt*sgravx
  :
7457:              !$OMP PARALLEL SHARED (sm3dh,sn3dh,sl3dh,sr3dh,se3dh
  :
7462:              !$OMP DO
7463: P------>           do iz=1,lz
7464: |+----->           do iy=1,ly
7465: ||V---->           do ix=3,lx-2
7466: |||     A             sm3dh(ix,iy,iz)=sm3dh(ix,iy,iz)
7467: |||               &                 +sdltthalfgx*sr3d(ix,iy,iz)
  :
7473: ||V----            enddo
7474: |+-----            enddo
7475: P------            enddo
7476:              !$OMP END DO NOWAIT
7477:              !$OMP END PARALLEL
  :
7498:              !$OMP PARALLEL SHARED (sm3dh,sn3dh,sl3dh,sr3dh,se3dh
  :
7502:              !$OMP DO
7503: P------>           do iz=1,lz
7504: |+----->           do iy=1,ly
7505: ||V---->           do ix=3,lx-2
7506: |||     A             sr3dhinv=1.0d0/sr3dh(ix,iy,iz)
7510: |||     A             su3dh(ix,iy,iz)=sr3dhinv*sm3dh(ix,iy,iz)
  :
7519: ||V----            enddo
7520: |+-----            enddo
7521: P------            enddo
7522:              !$OMP END DO NOWAIT
7523:              !$OMP END PARALLEL
```

図 5-5 ループ融合によるメモリアクセス低減前のコード

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
200	**13.111**(2.6)	65.557	36092.3	16377.9	99.26	129.0	13.102	0.002	0.005	**0.802**	**6.589**	【チューニング前】

図 5-6 ループ融合によるメモリアクセス低減前の FTRACE 情報

(2) チューニング内容

コンパイラによる自動ループ融合が行われない場合，ユーザチューニングによるループ融合が効果的である．チューニング後コードを図 5-7 に示す．連続する複数の DO ループは同じループ構造であり，一つの DO ループに修正することでコンパイラが最適化を行う．ループ融合後の配列

sm3dh に注目すると，演算で求めた結果をレジスタに格納しその後の演算ではレジスタ内のデータを参照することになるため，メモリアクセス回数はメモリストアの 1 回になり，メモリアクセス回数を削減することができる．表 5-1 にチューニング前後のメモリアクセス回数の一覧を示す．

```
!チューニング後
7858:                    sdltthalfgy=0.5d0*adltt*sgravy
7859:              !$OMP PARALLEL PRIVATE (ix,iy,iz,sr3dhinv)
7860:              !$OMP DO
7861: P------>           do iz=1,lz
7862: |+----->           do iy=1,ly
7863: ||V---->           do ix=3,lx-2
  :
7868: |||     A             sm3dh(ix,iy,iz)=sm3d(ix,iy,iz)
7869: |||               &             +sdltxinv*( du(ix-1,iy,iz)-du(ix,iy,iz) )
  :
7875: |||                   sm3dh(ix,iy,iz)=sm3dh(ix,iy,iz)
7876: |||               &                 +sdltthalfgx*sr3d(ix,iy,iz)
  :
7881: |||                   sr3dhinv=1.0d0/sr3dh(ix,iy,iz)
7882: |||                   su3dh(ix,iy,iz)=sr3dhinv*sm3dh(ix,iy,iz)
  :
7891: ||V----            enddo
7892: |+-----            enddo
7893: P------            enddo
7894:              !$OMP END DO NOWAIT
7895:              !$OMP END PARALLEL
```

図 5-7 ループ融合によるメモリアクセス低減後のコード

表 5-1 ループ融合によるメモリアクセス低減前後のメモリアクセス命令数

	チューニング前	チューニング後
メモリロード	36	16
メモリストア	17	9

(3) 性能分析

図 5-8 にチューニング前後の性能値を示す．

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CPU PORT	CONFLICT NETWORK	PROC.NAME
200	**13.111**(2.6)	65.557	36092.3	16377.9	99.26	129.0	13.102	0.002	0.005	0.802	**6.589**	【チューニング前】
200	**8.791**(1.8)	43.953	52951.2	24427.6	99.47	129.0	8.770	0.002	0.003	0.769	**3.745**	【チューニング後】

図 5-8 ループ融合によるメモリアクセス低減前後の FTRACE 情報

メモリアクセス回数が減少することでバンクコンフリクト時間が削減され，実行時間が 13.1 秒から 8.8 秒に短縮することができた．

5.3. 内側ループの展開によるメモリアクセスの削減

(1) チューニング方針

図 5-9 にチューニング前のコードを示す．最内側 n の DO ループの長さ(変数 nmax)は PARAMETER 文で 5 と宣言されており，ベクトル化の対象ループとなるが 121,122 行目の変数 vcx1，vcy2 はベクトル化の阻害要因となるため，コンパイラは最内側 n の DO ループでベクトル化を行わない．一つ外側 ic の DO ループの長さは 4 のため，コンパイラは更に外側 i の DO ループ

でベクトル化を行う. i の DO ループのベクトル化に伴い, ループ分割とループ入れ換えを行う. ループ分割では, i の DO ループのみに依存する演算と i, ic, n の DO ループに依存する演算に分割し, 演算結果を作業配列に格納する. 次に i, ic, n の DO ループの演算では, i の DO ループを内側に移動し事前に求めた作業配列の値を使用してベクトル処理する. このように外側の DO ループをベクトル化することで作業配列を用いたループ分割とループ入れ換えを行うためメモリアクセスの回数が増大する. 図 5-10 の FTRACE 情報では, 実行時間 267 秒の内バンクコンフリクト時間が 196 秒を占めるため, メモリアクセス回数を減らしてバンクコンフリクト時間を短縮することを検討する.

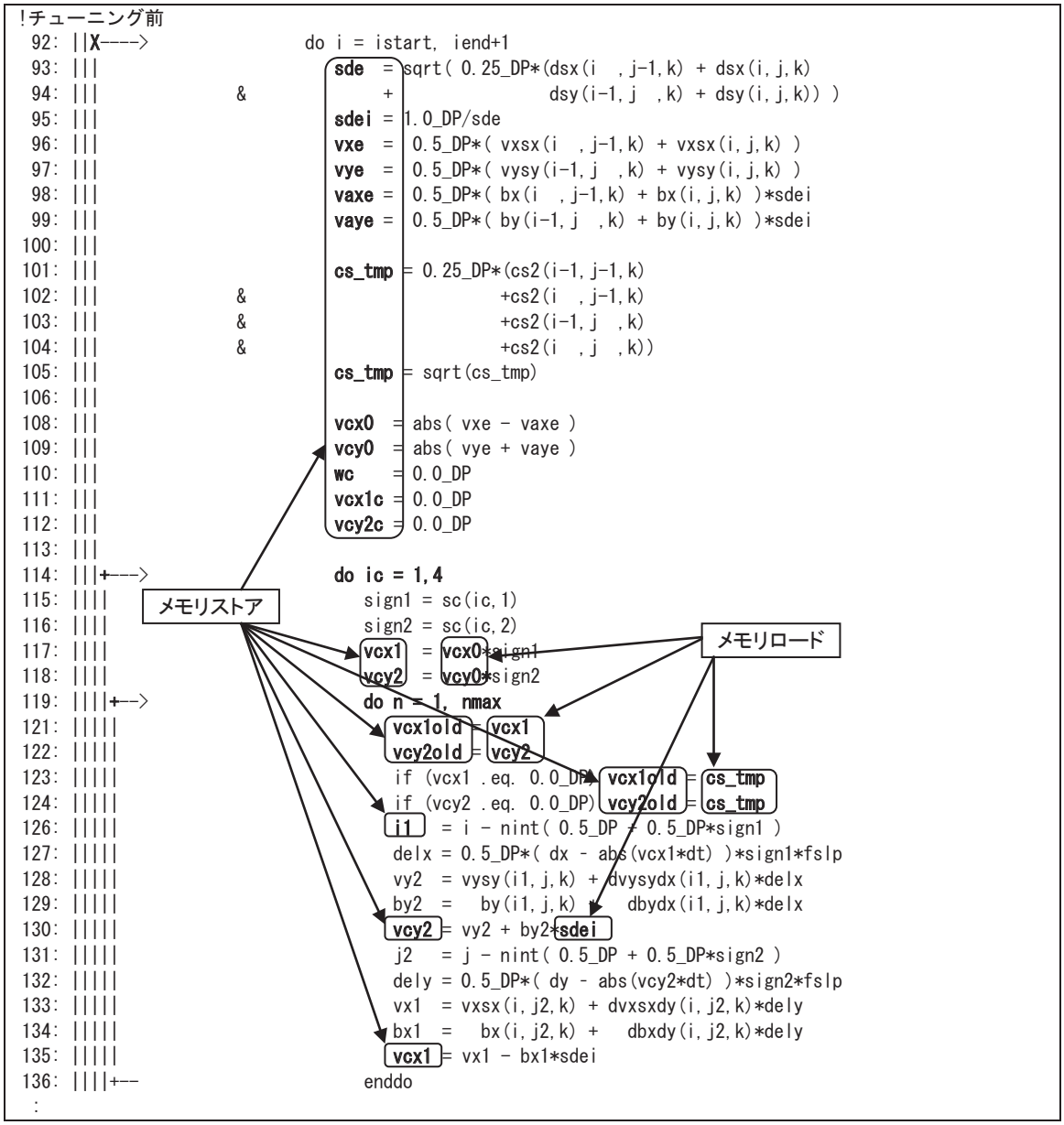

図 5-9　内側ループの展開によるメモリアクセス低減前のコード

```
 FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE   BANK CONFLICT   PROC.NAME
            TIME[sec](  %)    [msec]                   RATIO V.LEN     TIME    MISS    MISS CPU PORT  NETWORK

         50  266.912( 17.9)  5338.239  22400.9  6851.1 99.59 213.8   265.674   0.002   0.002  17.916  178.408 【チューニング前】
```

図 5-10 内側ループの展開によるメモリアクセス低減前の FTRACE 情報

(2) チューニング内容

内側の DO ループの長さがコンパイル時に判明しているため，EXPAND，UNROLL 指示行を挿入しループの展開を行う．内側ループを展開することで単純なベクトルループとなるため，ループ分割とループ入れ換えのための作業配列を使用する必要が無く，レジスタのデータで演算を行うことが可能になる．作業配列を使用しないためメモリアクセスの回数を削減することができる．図 5-11 にチューニング後コード，表 5-2 にチューニング前後のメモリアクセス回数の一覧を示す．

```
!チューニング後
 93: ||V---->               do i = istart, iend+1
 94: |||                       sde  = sqrt( 0.25_DP*(dsx(i  ,j-1,k) + dsx(i,j,k)
 95: |||             &              +              dsy(i-1,j  ,k) + dsy(i,j,k)) )
 96: |||                       sdei = 1.0_DP/sde
 97: |||                       vxe  =  0.5_DP*( vxsx(i  ,j-1,k) + vxsx(i,j,k) )
 98: |||                       vye  =  0.5_DP*( vysy(i-1,j  ,k) + vysy(i,j,k) )
 99: |||                       vaxe =  0.5_DP*( bx(i  ,j-1,k) + bx(i,j,k) )*sdei
100: |||                       vaye =  0.5_DP*( by(i-1,j  ,k) + by(i,j,k) )*sdei
101: |||
102: |||                       cs_tmp = 0.25_DP*(cs2(i-1,j-1,k)
103: |||             &                          +cs2(i  ,j-1,k)
104: |||             &                          +cs2(i-1,j  ,k)
105: |||             &                          +cs2(i  ,j  ,k))
106: |||                       cs_tmp = sqrt(cs_tmp)
107: |||
109: |||                       vcx0  = abs( vxe - vaxe )
110: |||                       vcy0  = abs( vye + vaye )
111: |||                       wc    = 0.0_DP
112: |||                       vcx1c = 0.0_DP
113: |||                       vcy2c = 0.0_DP
114: |||
115: |||         !cdir unroll=4
116: |||*--->                  do ic = 1,4
117: ||||                        sign1 = sc(ic,1)
118: ||||                        sign2 = sc(ic,2)
119: ||||                        vcx1  = vcx0*sign1
120: ||||                        vcy2  = vcy0*sign2
121: ||||        !cdir expand=nmax
122: ||||*-->                   do n = 1, nmax
124: |||||                        vcx1old = vcx1
125: |||||                        vcy2old = vcy2
126: |||||                        if (vcx1 .eq. 0.0_DP) vcx1old = cs_tmp
127: |||||                        if (vcy2 .eq. 0.0_DP) vcy2old = cs_tmp
129: |||||                        i1   = i - nint( 0.5_DP + 0.5_DP*sign1 )
130: |||||                        delx = 0.5_DP*( dx - abs(vcx1*dt) )*sign1*fslp
131: |||||                        vy2  = vysy(i1,j,k) + dvysydx(i1,j,k)*delx
132: |||||                        by2  =   by(i1,j,k) +   dbydx(i1,j,k)*delx
133: |||||                        vcy2 = vy2 + by2*sdei
134: |||||                        j2   = j - nint( 0.5_DP + 0.5_DP*sign2 )
135: |||||                        dely = 0.5_DP*( dy - abs(vcy2*dt) )*sign2*fslp
136: |||||                        vx1  = vxsx(i,j2,k) + dvxsxdy(i,j2,k)*dely
```

```
137: |||||                          bx1  =   bx(i,j2,k) +   dbxdy(i,j2,k)*dely
138: |||||                          vcx1 = vx1 - bx1*sdei
139: ||||*--                      enddo
 :
```

図 5-11　内側ループの展開によるメモリアクセス低減後のコード

表 5-2　内側ループの展開によるメモリアクセス低減前後のメモリアクセス命令数

	チューニング前	チューニング後
メモリロード	471	332
メモリストア	329	4
間接参照	160	0

(3)　性能分析

図 5-12 にチューニング前後の性能値を示す.

```
FREQUENCY  EXCLUSIVE        AVER.TIME     MOPS    MFLOPS V.OP  AVER.     VECTOR I-CACHE O-CACHE   BANK CONFLICT   PROC.NAME
           TIME[sec](  %)     [msec]                      RATIO V.LEN      TIME    MISS    MISS CPU PORT  NETWORK

      50  266.912( 17.9)  5338.239  22400.9   6851.1 99.59 213.8   265.674   0.002   0.002  17.916  178.408 【チューニング前】
      50   82.086(  8.5)  1641.720  46808.5  22759.7 99.46 214.4    70.789   3.235   0.002   0.086   14.105 【チューニング後】
```

図 5-12　内側ループの展開によるメモリアクセス低減前後の FTRACE 情報

メモリアクセス回数が減少することでバンクコンフリクト時間が短縮され, 実行時間が 266.9 秒から 82.1 秒に高速化することができた.

5.4. 複素数のバンク競合回避

(1)　チューニング方針

図 5-13 に FTRACE 情報を示す. FTRACE 情報では, 実行時間に対してバンクコンフリクトの占める割合が大きいことが分かる. 該当のサブルーチンでは複素数型配列を利用しているために飛びアクセスが発生していると考えられる. このため, 複素数を実数部と虚数部に分け実数型配列に保存し, メモリ上の連続する領域へのアクセスとすることにより, 速度低下やバンクコンフリクトを招く要因となりうる飛びアクセスの削減が期待できる.

```
FREQUENCY  EXCLUSIVE        AVER.TIME     MOPS    MFLOPS V.OP  AVER.     VECTOR I-CACHE O-CACHE   BANK CONFLICT   PROC.NAME
           TIME[sec](  %)     [msec]                      RATIO V.LEN      TIME    MISS    MISS CPU PORT  NETWORK

   44000 1453.723( 41.5)    33.039  31198.3  13481.2 99.62 255.9 1451.882   0.380   0.709 255.084  821.097 【チューニング前】
```

図 5-13　複素数のバンク競合回避前の FTRACE 情報

```
!チューニング前
   951: V------>            DO 101 JG=1,NXYZ
   952: V------       101   RHO2(JG)=(0.D0,0.D0)
    :
   955:             *VDIR NODEP(RHO1)
   956: V------>            DO 100 IG=1,NXYZ
   957: |                   JG=J2G(IG)
   958: V------       100   RHO1(JG)=P(IG)
    :
   991: V------>          do ig=1,nxyz
   992: |                 VG(ig)=VGG(ig)+Vloc(ig)
   993: V------           enddo
    :
  1009: V------>            DO 300 I=1,NXYZ
  1010: |                   fac=dt*dreal( vg(i) )
  1011: V------       300   RHO2(I)=dcmplx( dcos(fac),-dsin(fac) )*RHO1(I)
    :
  1028:             *VDIR NODEP(RHO2)
  1029: V------>            DO 110 IG=1,NXYZ
  1030: |                   JG=J2G(IG)
  1031: V------       110   P(IG)=RHO2(JG)
```

図 5-14 複素数のバンク競合回避前の前コード

(2) チューニング内容

図 5-14 にチューニング前の複素数型配列を利用したコード, 図 5-15 に最適化後の複素数を実数部と虚数部に分け実数型配列にしたチューニング後コードを示す. チューニング前コードでは倍精度複素数型を使用し配列 RHO1, RHO2 に演算を行っている. この該当箇所では倍精度実数型を使用し実数部は RHO1_R, RHO2_R とし, 虚数部は RHO1_I ,RHO2_I にそれぞれ分け演算を行い処理するようにする.

```
!チューニング後
   953: V------>            DO  JG=1,NXYZ
   954: |                     RHO2_R(JG)=real(0.D0)
   955: |                     RHO2_I(JG)=aimag((0.D0,0.D0))
   956: V------              ENDDO
    :
   968:             *CDIR NODEP
   969: V------>            DO  IG=1,NXYZ
   970: |                     JG=J2G(IG)
   971: |                     RHO1_R(JG)=dble(P(IG))
   972: |                     RHO1_I(JG)=aimag(P(IG))
   973: V------              ENDDO
    :
   984: V------>          do I=1,nxyz
   985: |                   VG_R(I)=VGG(I)+Vloc(I)
   986: |                   fac=dt*VG_R(I)
   987: |                   RHO2_R(I)=(dcos(fac)*RHO1_R(I))-((-dsin(fac))*RHO1_I(I))
   988: |                   RHO2_I(I)=(dcos(fac)*RHO1_I(I))-((dsin(fac))*RHO1_R(I))
   989: V------           enddo
    :
   995:             *CDIR NODEP
   996: V------>            DO  IG=1,NXYZ
   997: |                     JG=J2G(IG)
   998: |                     P(IG)=dcmplx(RHO2_R(JG),RHO2_I(JG))
   999: |                     VG(IG)=dcmplx(VG_R(IG),0.0d0)
  1000: V------              ENDDO
```

図 5-15 複素数のバンク競合回避後のコード

(3) 性能分析

図 5-16 は, チューニング前後の性能値を示す.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER. TIME [msec]	MOPS	MFLOPS	V. OP RATIO	AVER. V. LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC. NAME
44000	**1453.723**(41.5)	33.039	31198.3	13481.2	99.62	255.9	1451.882	0.380	0.709	255.084	821.097	【チューニング前】
44000	**996.058**(34.9)	22.638	45680.0	19675.5	99.63	255.9	993.704	0.676	1.039	234.856	466.970	【チューニング後】

図 5-16 複素数のバンク競合回避前後の FTRACE 情報

メモリ上の連続する領域へのアクセスすることにより, 不連続なメモリアクセスが軽減され, 実行時間が 1,453 秒から 996 秒に短縮することができた.

6. ベクトル演算の効率化の事例

6.1. 総和演算の効率化

(1) チューニング方針

ベクトルループ内の総和演算は一回前の繰り返しの結果を参照するためベクトル化の阻害要因となる. しかし, コンパイラは総和演算を特別なパターンであることを認識し, 専用のベクトル命令(総和型マクロ演算)を用いることによりベクトル化を行う. 図 6-1 はチューニング前のコードである. 配列 Wn は i の次元を持たないため i の DO ループをベクトル化すると, コンパイラは総和型マクロ演算を用いてベクトル化を行う. この総和型マクロ演算により実行を高速化できるが, 総和演算を含む多重ループの場合, ループを入れ替え, 総和型マクロ演算をベクトル演算にすることによりさらなる高速化が可能となる場合がある. このような事例を以下に示す.

```
!チューニング前
11: +------>       do j=1,ny
12: |V----->       do i=1,nx
13: ||
14: ||              Wn(j) = Wn(j) + Wm(i,j)
15: ||
16: |V-----        end do
17: +-------       end do
```

図 6-1 総和演算の効率化前のコード

(2) チューニング内容

図 6-2 にループ入れ換え後のコードを示す. ループ入れ換えにより j の DO ループでベクトル化することで配列 Wm のメモリアクセスがストライドになることに注意されたい.

```
!チューニング後
12: +------>       do i=1,nx
13: |V----->       do j=1,ny
14: ||
15: ||              Wn(j) = Wn(j) + Wm(i,j)
16: ||
17: |V-----        end do
18: +-------       end do
```

図 6-2 総和演算の効率化後のコード

(3) 性能分析

図 6-3 にチューニング前後の性能値を示す.

```
 FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE   BANK CONFLICT   PROC.NAME
           TIME[sec](  %)     [msec]                    RATIO V.LEN     TIME    MISS    MISS CPU PORT  NETWORK

         1  232.086( 71.8)232086.438  17813.0  8620.3 99.30 173.6  232.083   0.001   0.002    0.000   32.327 【チューニング前】
         1   91.148( 28.2) 91148.138  36054.1 20772.5 99.11 191.0   91.147   0.000   0.001   27.536    4.160 【チューニング後】
```

図 6-3 総和演算の効率化前後の FTRACE 情報

チューニングを行うことで実効性能が向上し, 実行時間が 232.1 秒から 91.1 秒に高速化することができた.

6.2. IF 文の括り出しによる演算数の削減

(1) チューニング方針

ベクトルループ内に IF 文がある場合, SX-9 では IF 文の真偽に関わらず全ての演算を行い最後に IF 文の真にあたる結果のみを採用する方法でベクトル処理を行う. 図 6-4 にチューニング前のコードを示す.

```
!チューニング前
12: +------>       do j=1,my
13: |V----->       do i=1,mx
14: ||               if(itbl(i)==1)then
15: ||                   Wn(i,j) = Wn(i,j) + Wm(i,j)
16: ||               end if
17: |V-----        end do
18: +------        end do
```

図 6-4 IF 文の括り出しによる演算数削減前のコード

(2) チューニング内容

図 6-5 にチューニング後のコードを示す. IF 文の判定式である配列 itbl はベクトル化対象である i の DO ループの次元しか持たないため, IF 文を含めたループ入れ換えを行うことでベクトルループ内に IF 文の処理が無くなる. ベクトル計算機では, ベクトルループ内に IF 文がある場合, 条件式の真偽に関わらず両方の演算を行い, 最後に条件式が真となる場合の結果を選択する. 両方の演算を行うため演算数は増加するが, ベクトル演算を行うことにより性能が向上する. ループ入れ換えを行うことにより, ベクトル演算の効率を低下させることなく演算数を削減し, 実行時間を短縮することができる場合がある. このような高速化の事例を以下に示す. なお, ループの入れ換えにより, 配列 Wn, Wm のメモリアクセスがストライドとなるため, メモリアクセスの観点から性能が低下する可能性があり, 実際に高速化できるかについては確認が必要である.

```
!チューニング後
12: +------>       do i=1,mx
13: |                if(itbl(i)==1)then
14: |V----->       do j=1,my
15: ||                   Wn(i,j) = Wn(i,j) + Wm(i,j)
16: |V-----        end do
17: |                end if
18: +------        end do
```

図 6-5 IF 文の括り出しによる演算数削減後のコード

(3) 性能分析

図 6-6 にチューニング前後の性能値を示す.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
1	**98.349**(68.2)	98349.331	36691.7	**20319.8**	98.99	191.0	98.348	0.000	0.001	22.750	4.486	【チューニング前】
1	**45.849**(31.8)	45848.683	36026.1	**20756.3**	99.11	191.0	45.847	0.000	0.001	13.683	2.121	【チューニング後】

図 6-6　IF 文の括り出しによる演算数削減前後の FTRACE 情報

ベクトル演算量(実行時間×実効性能)が約 1/2 となるため, 実行時間が 98.3 秒から 45.9 秒に短縮することができた.

6.3. ライブラリの置き換えによる高速化

(1) チューニング方針

SX-9 では, HPC 用に高度に最適化された数学ライブラリ集の Mathkeisan が利用できる. BLAS/LAPACK も最適化されており, プログラムの高速化が容易に行える.

図 6-7 にチューニング前のコードを示す. このコードでは, LAPACK のサブルーチン zheev が呼び出されている. 図 6-8 のチューニング前の性能情報を確認すると, ベクトル化率が 98.17%となっている. zheev は QR 法を用いてエルミート行列の固有値・固有ベクトルを求めるが, 分割統治法でこれらを求めるライブラリ zheevd に置き換えることでベクトル化率の向上と計算時間の短縮を図る.

なお, 分割統治法は QR 法より高速と知られているが, QR 法の演算量が行列の次数の 3 乗のオーダーであるのに対し, 分割統治法の演算量は 2 乗から 3 乗のオーダーとばらつきがあることから, 場合によっては, 有意な性能差がみられないケースもありうる. また, 必要となるメモリ領域の大きさについては, QR 法は次数の 1 乗のオーダーであるのに対し, 分割統治法は 2 乗のオーダーとなる点も考慮する必要がある.

```
!チューニング前
983:        call zheev('V','U',nb*2,v2,nb*2,eig2,work2,2*(nb*2)-1,rwork2,  info)
```

図 6-7 ライブラリ置き換え前のコード

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
4	78.111(5.0)	19527.870	7546.6	4231.4	98.17	210.4	27.766	0.022	23.009	3.657	18.205	【チューニング前】

図 6-8 ライブラリ置き換え前の FTRACE 情報

(2) チューニング内容

図 6-9 にチューニング後のコードを示す. チューニング前のコードで呼び出されていた LAPACK のサブルーチン zheev を zheevd に置き換えている. zheev と zheevd は呼び出しの際の引数が異なり, 作業用の配列も 2 つ増える. 作業配列のサイズは計算前に事前に一度 zheevd を呼び出すことで, 計算に最適なサイズを取得することができる. 取得したサイズの作業配列を確保し, zheevd の呼び出す.

```
!チューニング後
184:                    lwork2= -1
185:                    lrwork2= -1
186:                    liwork2= -1
187:                    call zheevd('V','U',nb*2,v2,nb*2,eig2,work2,lwork2,rwork2,
188:           &             lrwork2,iwork2,liwork2,info)
189:                    lwork2=work2(1)
190:                    lrwork2=rwork2(1)
191:                    liwork2=iwork2(1)
192:                    deallocate (work2,rwork2,iwork2)
193:                    allocate (work2(lwork2),rwork2(lrwork2),iwork2(liwork2))
 :
986:              call zheevd('V','U',nb*2,v2,nb*2,eig2,work2,lwork2,rwork2,
987:           &           lrwork2,iwork2,liwork2,info)
```

図 6-9 ライブラリ置き換え後のコード

(3) 性能分析

図 6-10 はチューニング後の性能情報である. 適切なライブラリを選択することでベクトル化率が 98.17%から 99.38%へ向上し, 78.11 秒から 6.954 秒に短縮することができた. また, zheev から zheevd に置換することによる計算結果への影響もみられなかった.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
4	78.111(5.0)	19527.870	7546.6	4231.4	98.17	210.4	27.766	0.022	23.009	3.657	18.205	【チューニング前】
4	6.954(0.5)	1738.465	53869	31038.0	99.38	221.5	6.729	0.014	0.059	1.026	2.812	【チューニング後】

図 6-10 ライブラリ置き換え前後の FTRACE 情報

ここではライブラリルーチンの置き換えによる高速化の事例を挙げたが, 置換する際には, 置換によるメリット・デメリットを把握した上で置換を行い, 置換後の計算結果の妥当性も必ず確認する必要がある.

6.4. ファイルアクセスの高速化

(1) チューニング方針

図 6-11 に FTRACE 情報を示す. FTRACE 情報では, ベクトル化率が 1.2%となっている. 詳細な解析により, 図 6-12 に DO ループの中でファイルからデータを読み込んだ直後, 読み込んだデータに対して演算が実行されていることが分かった. そこで, ファイルの読み込みと演算を別々に処理するよう最適化を行う.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
1	219.190(12.5)	219189.905	469.7	4.1	1.20	248.1	0.082	34.416	5.627	0.395	0.057	【チューニング前】

図 6-11 ファイルアクセス高速化前の FTRACE 情報

```
!チューニング前
  2776: +------>          DO 450 IK = 1, NUMK
   :    |
  2783: |                 if ( my_rank.eq.0 ) then
  2784: |+----->           do icpu=0,ncpu
  2785: ||                 nbleng=nend(icpu)-nbegin(icpu)+1
  2786: ||+---->           do 451 ib=1,nbleng
  2787: |||                read(22)( rho1(ig),rho2(ig),ig=1,nxyz )
  2788: |||V--->            do ig=1,nxyz
  2789: ||||                coef0(ig,ib,ik)=dcmplx( rho1(ig),rho2(ig) )
  2790: |||V---             enddo
  2791: ||+----       451   continue
  2792: ||                 if ( icpu.eq.0 ) then
  2793: ||+---->            do ib=1,nbleng
  2794: |||V--->             do ig=1,nxyz
  2795: ||||                 coef(ig,ib,ik)=coef0(ig,ib,ik)
  2796: |||V---              enddo
  2797: ||+----             enddo
  2798: ||                 else
  2799: ||                  call MPI_Send(coef0(1,1,ik),nxyz*nbleng,
  2800: ||                &  MPI_DOUBLE_COMPLEX,icpu,tag,MPI_COMM_WORLD,ierr)
  2801: ||                 endif
  2802: ||
  2803: |+-----            enddo  ! end of icpu loop
  2804: |                 else
  2805: |                  nbleng=nend(my_rank)-nbegin(my_rank)+1
  2806: |                  call MPI_Recv(coef0(1,1,ik),nxyz*nbleng
  2807: |                & ,MPI_DOUBLE_COMPLEX,0,tag,MPI_COMM_WORLD,status,ierr)
  2808: |+----->           do ib=1,nbleng
  2809: ||V---->            do ig=1,nxyz
  2810: |||                 coef(ig,ib,ik)=coef0(ig,ib,ik)
  2811: ||V----             enddo
  2812: |+-----            enddo
  2813: |                 endif
  2814: +------       450 CONTINUE      ! ik roop
```

図 6-12 ファイルアクセスの高速化前のコード

(2) チューニング内容

図 6-12 にチューニング前コード, 図 6-13 にチューニング後コードを示す. チューニング前コードでは read(22) が DO ループの中にあり, その中でさらに演算を行っている. この該当箇所に対して read(22) を DO ループの外へ移動し, 演算に関連する部分の配列を 1 次元配列から 2 次元配列に変更しファイルの読み込みと演算を別々に処理するようにする.

```
!チューニング後
  2776: +------>          DO 450 IK = 1, NUMK
   :    |
  2783: |                 if ( my_rank.eq.0 ) then
  2784: |+----->            do ib=1,nend(ncpu)
  2785: ||                    read(22)(tmp_22(ig,ib),ig=1,nxyz)
  2786: |+-----             enddo
  2787: |                   icnt=0
  2788: |+----->            do icpu=0,ncpu
  2789: ||                    nbleng=nend(icpu)-nbegin(icpu)+1
  2790: ||+---->              do ib=1,nbleng
  2791: |||V--->                do ig=1,nxyz
  2792: ||||                     coef0(ig,ib,ik)=tmp_22(ig,ib+icnt)
  2793: |||V---                 enddo
  2794: ||+----                enddo
  2795: ||                     icnt=icnt+nbleng
  2796: ||                     if ( icpu.eq.0 ) then
  2797: ||+---->                do ib=1,nbleng
  2798: |||V--->                  do ig=1,nxyz
  2799: ||||                       coef(ig,ib,ik)=coef0(ig,ib,ik)
  2800: |||V---                   enddo
  2801: ||+----                  enddo
  2802: ||
  2803: ||                     else
  2804: ||                     call MPI_Send(coef0(1,1,ik),nxyz*nbleng,
  2805: ||                &      MPI_DOUBLE_COMPLEX,icpu,tag,MPI_COMM_WORLD,ierr)
  2806: ||
  2807: ||                     endif
  2808: |+-----              enddo  ! end of icpu loop
   :    |
  2811: |                   else
  2812: |                   nbleng=nend(my_rank)-nbegin(my_rank)+1
  2813: |                   call MPI_Recv(coef0(1,1,ik),nxyz*nbleng,
  2814: |                &    MPI_DOUBLE_COMPLEX,0,tag,MPI_COMM_WORLD,status,ierr)
  2815: |+----->            do ib=1,nbleng
  2816: ||V---->              do ig=1,nxyz
  2817: |||                     coef(ig,ib,ik)=coef0(ig,ib,ik)
  2818: ||V----               enddo
  2819: |+-----              enddo
  2820: |                   endif
  2821: +------       450 CONTINUE      ! ik roop
```

図 6-13 ファイルアクセスの高速化後のコード

(3) 性能分析

図 6-14 にチューニング前後の性能値を示す.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
1	219.190(12.5)	219189.905	469.7	4.1	**1.20**	248.1	0.082	34.416	5.627	0.395	0.057	【チューニング前】
1	0.095(0.0)	94.761	13163.1	0.0	**98.99**	248.0	0.087	0.001	0.001	0.017	0.061	【チューニング後】

図 6-14 ファイルアクセスの高速化前後の FTRACE 情報

ファイルの読み込みと演算を別々に処理することにより, ベクトル化率が 1.2%から 98.99%に改善したことで実行時間が 219.190 秒から 0.095 秒に短縮することができた.