[高速化推進研究活動報告　第３号]より

４　ＳＸ－７高速化技法

スーパーコンピューティング研究部　岡部公起
システム管理係　伊藤英一
日本電気株式会社　撫佐昭裕，石井繭子，吉村健二，金野浩伸
NEC システムテクノロジー株式会社　曽我隆

４．１　ＳＸ－７の特長

SX-7 は，1 ノードあたり 32 個の CPU，256GB の主記憶を搭載し，282.5GFLOPS のベクトル演算性能を有したベクトル型スーパーコンピュータである．SX-7 の CPU はベクトルユニットとスカラユニットからなり，主な特長は以下のとおりである．

① ベクトルユニットは, 8 セットのベクトル演算パイプライン（マスク演算, 論理演算, 乗算，加算／シフト演算，除算）を有し，1CPU あたり 8.83GFLOPS の性能を実現している．
② 共有メモリ方式のアーキテクチャにより，256GB の大規模な主記憶を 32 個の CPU で共有することができる．
③ 主記憶と CPU 間のデータ転送は，1CPU あたり 35.32GB/S を実現している．
④ プログラムを高速実行するため FORTRAN と C コンパイラは, 自動ベクトル化機能と自動並列化機能を有している．
⑤ プログラムの性能解析を行うツールを有している．
⑥ SX-7 向けに高速化した数値計算ライブラリ ASL/SX を有している．

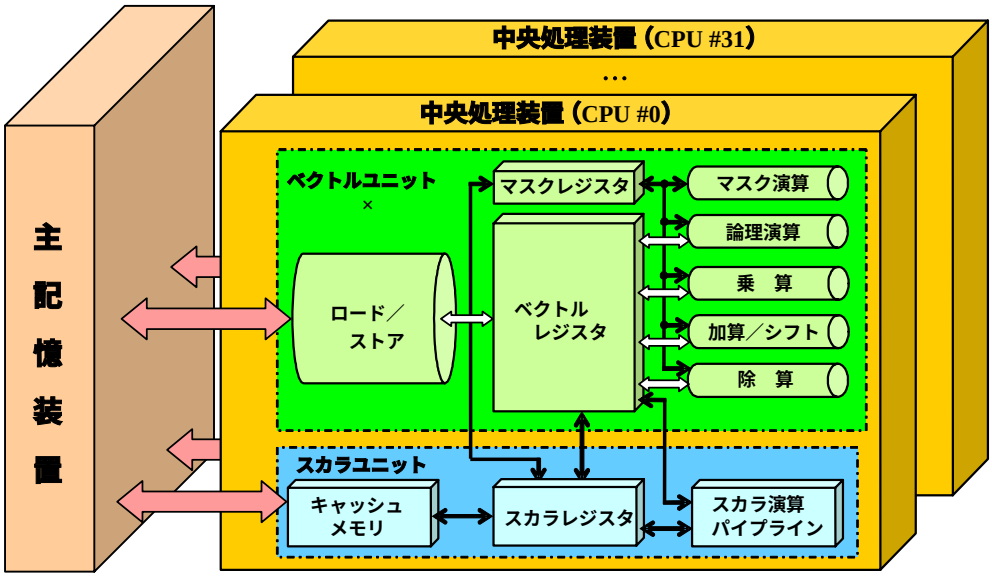

図 4.1.1　CPU の内部構造

表4.1.1　SX-7ノードの主要諸元

項目			SX-7
理論最大演算性能			282.56GFLOPS
CPU数			32
CPU	レジスタ	ベクトルレジスタ	144KB
		ベクトルマスクレジスタ	256bit×16
		スカラレジスタ	64bit×128
	データ形式	固定小数点	32/64bit
		浮動小数点	32/64/128*bit
			IEEE
		論理	64bit
	ベクトル演算パイプライン		5種類×8セット
	スカラ演算パイプライン		1セット
	キャッシュ		命令：64KB
			オペランド：64KB
主記憶装置	容量		256GB
	最大データ転送能力		1130.24GB/S

* 128bitはスカラ命令のみサポート

４．２　ベクトル処理

SX-7 における高速演算機能としてベクトル処理がある．SX-7 では，FORTRAN，C コンパイラの自動ベクトル化機能によって，ユーザは意識することなくベクトル処理を利用することができる．

科学技術計算プログラムの多くは，特定の do ループ(for ループ)に実行の大部分が集中している．しかも，このループは配列データ（ベクトル）に同一演算を繰り返し実行させることが多い．この規則性に着目して高速化を図った方式がベクトル処理である．ベクトル処理は，do ループによる配列データへの繰り返し演算を，ベクトルユニットを用いて一括に実行するものである．一方，このベクトル処理に対してスカラ処理といわれる演算は，一組のデータを逐次的に実行していくものである．

SX-7 のベクトルユニットは 5 種類のベクトルパイプライン（マスク演算，論理演算，乗算，加算／シフト演算，除算）からなり，データの依存関係がない場合，これらベクトルパイプラインは 5 種類同時に実行することができる．また，各ベクトルパイプラインは 8 セット用意され，SIMD(Single Instruction Multiple Data)型の並列処理を行っている．ベクトルユニットのこれらの動作が SX-7 の高速演算を実現しているのである．

（1）ベクトル化率

プログラムを高速に実行するためには，SX-7 のベクトルユニットをできるだけ多く利用する必要がある．図 4.2.1 に示すようにプログラムのベクトル処理可能部分を増やすことによって，プログラムの実行性能が向上し，実行時間を短縮することができる．

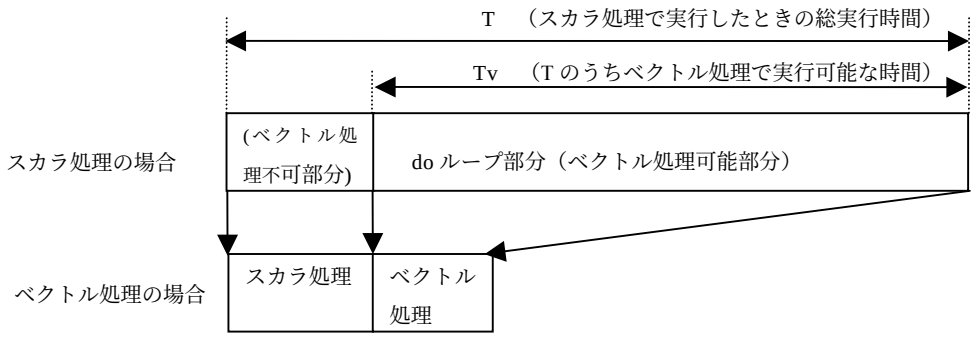

図 4.2.1　ベクトル処理による実行時間短縮

ベクトル処理を行ったときの性能向上倍率はベクトル化率から求めることができる（アムダールの法則）．プログラム内におけるベクトル処理の割合をベクトル化率と呼び，図 4.2.1 中の Tv と T を用いて，

$$\text{ベクトル化率}\quad \alpha = Tv / T$$

である．理想的な性能向上倍率 P は，次のように表される．

$$P = 1 / ((1-\alpha) + \alpha / \beta)$$

ここで β はスカラ性能とベクトル性能の比である．図 4.2.2 は，β を 50 と仮定した時の性能倍率である．この図より，プログラムを高速に実行するためには，ベクトル化率が 95%を超える必要がある．

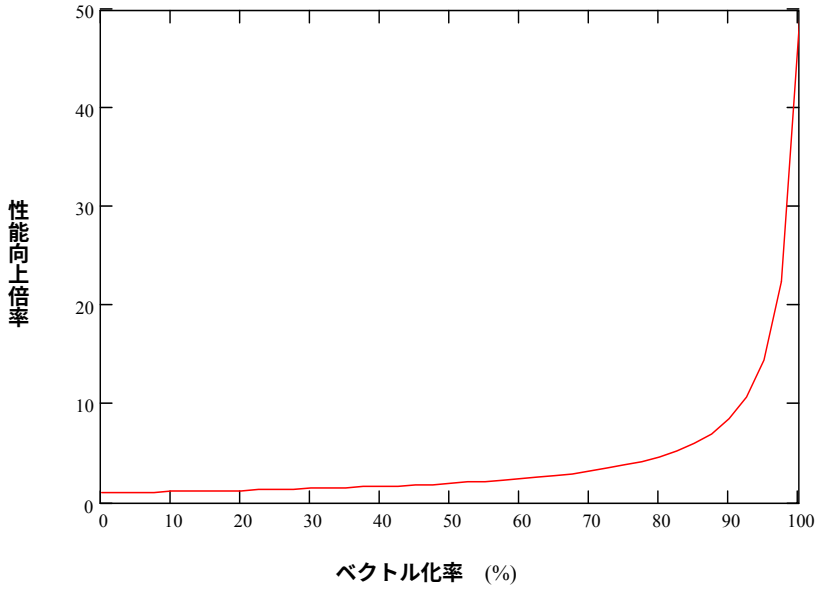

図 4.2.2　ベクトル化率と性能

（2）ベクトル長

ベクトル処理において SX-7 のベクトルユニットを効率よく利用するためには，do ループでのベクトル長（do ループのループ回数）を大きくする必要がある．ベクトルユニットでの演算には立ち上がり時間と呼ばれる演算準備時間が存在する．そのためベクトル長が小さい場合，立ち上がり時間の影響が顕著になりベクトル処理による高速化の効果を打ち消してしまう．

図 4.2.3 にベクトル長と実行時間の概念図を示す．ベクトル処理とスカラ処理とで実行時間が等しくなるベクトル長を交差ループ長と呼び，SX-7 では 5～10 程度である．図 4.2.3 より，ベクトル長が大きくなった場合，ベクトル処理はスカラ処理に比べ実行時間の延びが緩やかであることがわかる．このことより，ベクトル処理を主体とした SX-7 は大規模な計算に適しているのである．

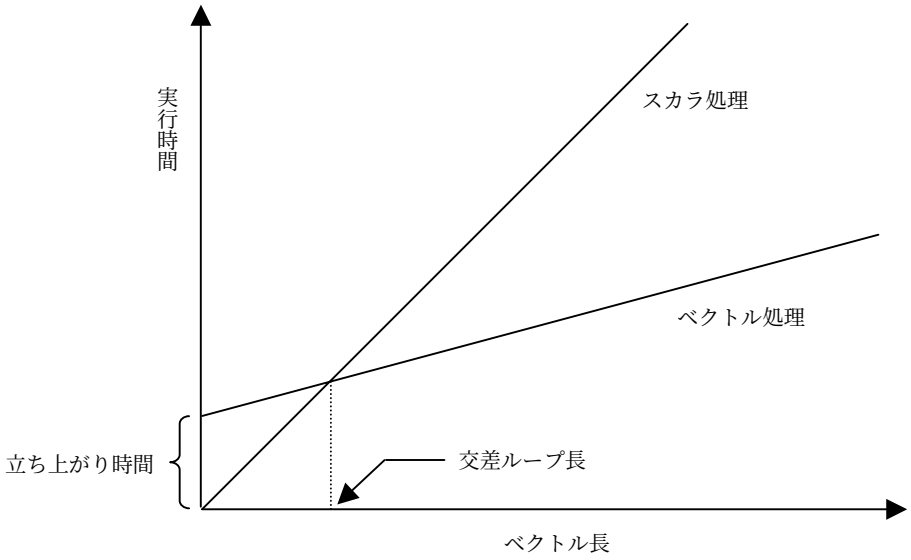

図 4.2.3　ベクトル長と実行時間

（3）メモリアクセス

SX-7 の高速演算は，主記憶装置からベクトルパイプラインへの高いデータ転送能力で支えられている．SX-7 では 1 CPU あたり 35.32GB/S，ノードあたり 1,130.24GB/S の転送能力を有している．この転送能力は主記憶装置に配置されている 16,384 個のバンクが効率的に動作することで実現している．

主記憶装置からベクトルパイプラインへのデータ転送において，特定のバンクだけに転送が集中した場合をバンクコンフリクトと呼び，ベクトルパイプラインへのデータ供給が滞り，ベクトル処理の遅延が発生する．バンクコンフリクトの原因は，配列の大きさや do ループ内の処理の仕方による．もし，この現象に該当した場合には 4.5 章のチューニング事例を参考にして改善をして頂きたい．

４．３　並列処理

SX-7 における高速演算機能として並列処理がある．SX-7 では FORTRAN，C コンパイラの自動並列化機能によって，1 ノード 32CPU までの並列処理が可能である．ここでは高速演算を目的とした並列処理の概要を説明する．

並列化機能は一つのプログラムを複数の処理単位に分割し，それぞれを異なる CPU で同時実行することによってプログラムの実行時間（経過時間）を短縮するものである．図 4.3.1 は並列実行可能な do ループを分割し，4CPU を用いて経過時間の短縮を行っている概念図である．

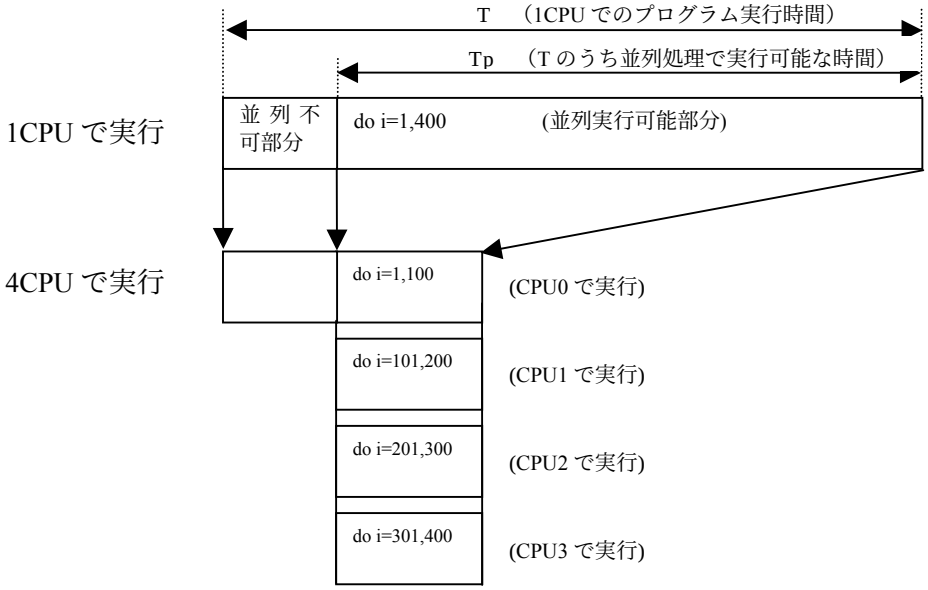

図 4.3.1　並列処理による実行時間短縮

（1）並列化率

並列処理においてプログラムをより短時間に処理するためには，図 4.3.1 に示した並列実行可能部分を増やす必要がある．

並列処理を行ったときの性能向上倍率は並列化率から求めることができる（アムダールの法則）．プログラム内における並列処理可能部分の割合を並列化率と呼び，図 4.3.1 中の T，Tp を用いて

$$並列化率　\alpha = Tp / T$$

である．並列化率 α のプログラムを n 台の CPU で演算を行ったときの理想的な性能向上倍率 P は，次のように表される．

$$P = 1 / ((1-\alpha) + \alpha / n)$$

図 4.3.2 に CPU8 台, 16 台, 32 台の性能向上倍率を示す. 並列性能を向上させるためには, 並列化率 95%以上が必要なことがわかる．また，並列化率が低い場合には，CPU 台数を増やしても性能が変わらないこともわかる．

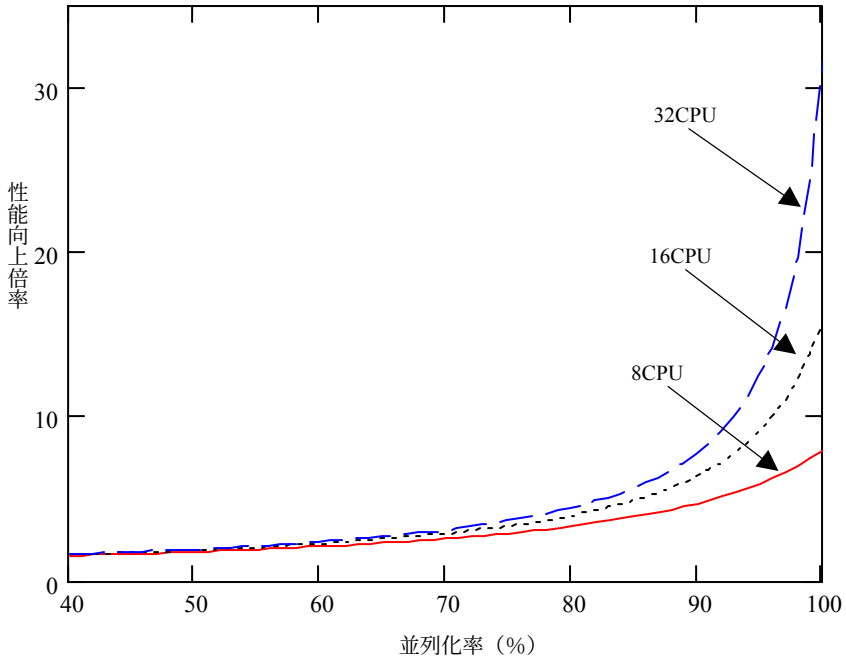

図 4.3.2　並列化率と性能倍率

（2）粒度

並列処理では，並列実行するための制御処理（タスク生成やバリヤ同期など）の時間が必要になる．この時間をオーバヘッドという．図 4.3.3 は並列処理の粒度とオーバヘッドの概念図である．図の a，b は，それぞれ同じ演算量を異なった粒度（1 つの並列単位の演算量）で並列化した場合の例である．図の a は，粒度が大きい場合であり，オーバヘッドが少ない．b は粒度が小さく，相対的にオーバヘッドが増加している．a，b を比較した場合，a は b よりオーバヘッドが少ない分，実行時間が短くなっている．

並列処理では，粒度が小さい場合，並列化を行ってもオーバヘッドにより速くならないことがある．並列処理のためには，粒度を大きくし，オーバヘッドを削減することが必要である．粒度を大きくするためには，

- 並列化する do ループに多くの演算を集める
- 多重 do ループの一番外側で並列化を行う

などの工夫が必要である．また，粒度の小さな do ループは並列化をしないなどの処置も必要である．

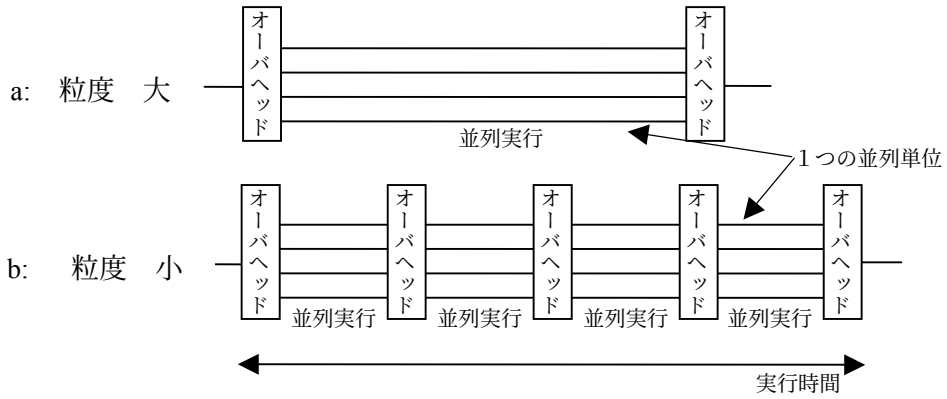

図 4.3.3 粒度とオーバヘッド

（3）負荷バランス

並列処理の効率を上げるためには，それぞれの CPU で実行する演算量（負荷バランス）が均等である必要がある．図 4.3.4 は負荷バランスの概念図である．図の a，b は，それぞれ同じ演算量を異なった負荷バランスで並列化した場合の例である．図の a は，並列処理を行う各 CPU に演算が均等に割り付けられた例で，効率的な並列処理を行っている．b は，CPU ごとにばらつきがあるので実行時間が長くなっている．

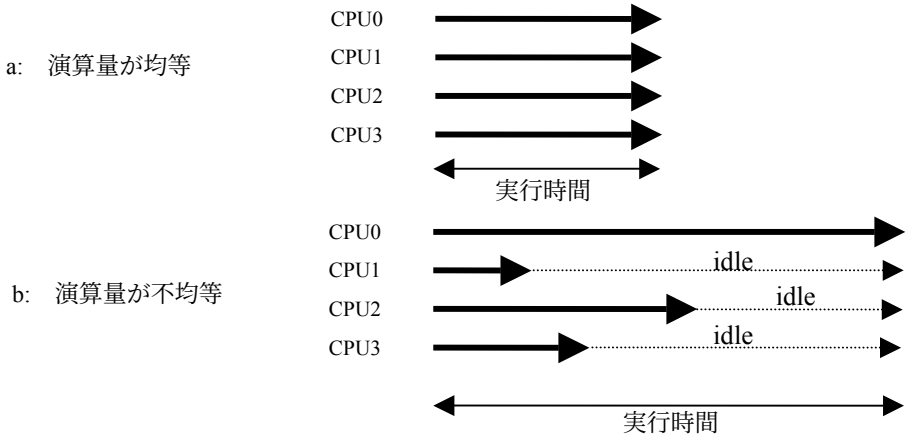

図 3.3.4 負荷バランス

SX-7 の FORTRAN，C コンパイラは， do ループの負荷バランスを調整するオプション（for=整数値，by=整数値）をサポートしている．コンパイラの既定値は，for=（使用できる CPU 数）である．図 4.3.5 に，for=4 と by=4 の例を示す．for=4 では，ループ長 100 の do ループが，ループ長 25 の do ループ 4 個に分割される．by=4 では，ループ長 100 の do ループが，ループ長 4 の do ループ 25 個に分割される．

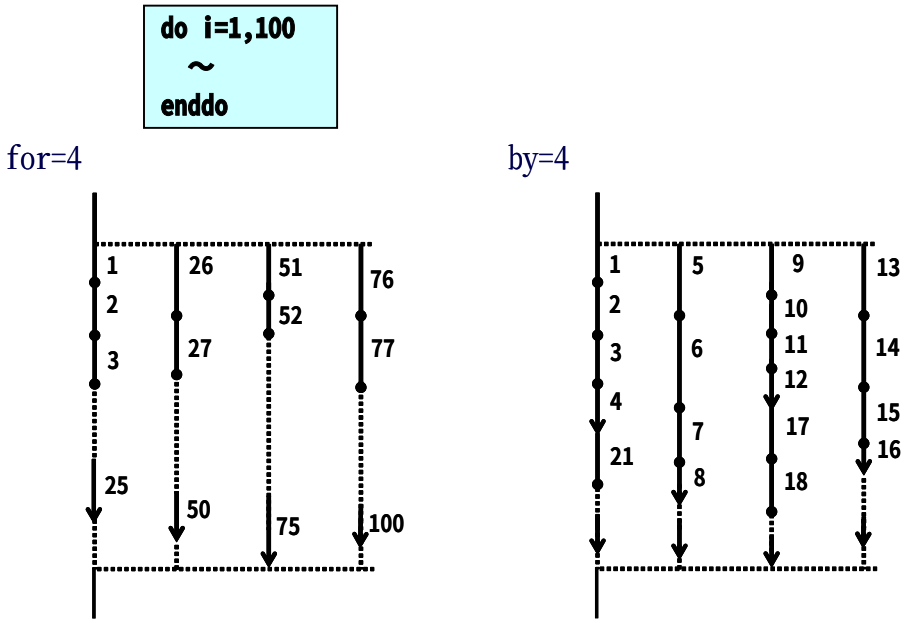

図 4.3.5　ループ長 100 のループを分割する場合

４．４　プログラムの解析

SX-7 のコンパイラは様々な最適化機能を有しており，SX-7 の性能を引き出すオブジェクトの生成を行う．しかし，プログラムによっては最適化が十分に行えない場合がある．そのような場合でも利用者が少しの工夫(チューニング)を行うだけで，高い性能を引き出せる可能性がある．ここでは，プログラム解析を行うために利用できる「編集リスト」，「プログラム情報」，および「ftrace 情報」を説明する．

（1）編集リスト

プログラムのどの部分がベクトル化，並列化されているかを知るためには，編集リストを採取する．編集リストは，コンパイル時にコンパイルオプション「-R5」を指定した場合に，「ソースファイル名.L」というファイル名で作成される．図 4.4.1 に編集リストの出力例を示す．

```
FILE NAME: list.f
PROGRAM NAME: sub
FORMAT LIST

   ①      ②     ③                ④
 LINE    LOOP        FORTRAN STATEMENT

   35:                      subroutine sub(a,b,c,d,x,z,ix)
   36:                      real,dimension(100)::a,b,c,d,x,y,z
   37:                      integer ix(100)
   38: V------>             do I=1,100
   39: |       I               call func(x,a,b,I)
   40: |                         y(I)=c(I) + d(I)
   41: |       S                 z(ix(I))=z(ix(I))+x(I)+y(I)
   42: V------              enddo
   43:                      end
```

図 4.4.1　編集リストの出力例

①　外部行番号
②　ループ情報
③　スカラ情報，インライン展開情報
④　原始プログラム

以下に主なループ情報，スカラ情報，インライン展開情報の出力イメージを示す．

- do ループ全体がベクトル化される場合

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

ベクトル化される do ループに「V」が表示される．

- do ループが部分的にベクトル化される場合

```
V------>          do i=1,100
|                    a(i)=b(i)+c(i)
|       S            z(ix(i))=z(ix(i))+x(i)+y(i)
V------           enddo
```

ベクトル化できない行（スカラ処理される行）には「S」が表示される．

- ベクトル化されない場合

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

ベクトル化されない do ループには「+」が表示される．

- 副プログラムの呼び出しがインライン展開される場合

```
        I         call sub
```

インライン展開される副プログラムがある行には「I」が表示される．

- 多重 do ループが一重化される場合

```
W------>           do j=1,100
|*----->           do i=1,100
||                   d(i,j)=d(i,j)+b(i,j)+c(i,j)
|*-----             enddo
W------            enddo
```

一重化される do ループの外側ループには「W」，内側ループには「*」が表示される．

- do ループが並列化される場合

```
P------>          do i=1,100
|                    :
|                    :
P------           enddo
```

並列化される do ループには「P」が表示される．

- 多重 do ループにおいて，do ループの入れ替えが行われる場合

```
X ------>           do i=1,100
|+----->           do j=1,20
||                   d(i,j)=d(i,j)+b(i,j)+c(i,j)
|+-----             enddo
X ------            enddo
```

do ループの入れ替えが行われる場合，ベクトル化される do ループに「X」が表示され，ベクトル化されない do ループには「+」が表示される．

（2）プログラム情報（Program Information）

プログラム全体の性能を得るには，プログラム情報(以下 Proginf)を採取する．本センターでは Proginf の採取を行う設定になっているので，プログラムを実行すると標準エラー出力に Proginf が出力される．これは，ハードウェア側で計測している情報を整理して出力されるだけなので，オーバヘッドはない． 図 4.4.2 に Proginf の出力例を示す．

```
******  プログラム 情報  ******
経過時間（秒）             :      6.774000
ユーザ時間（秒）           :      4.911325
システム時間（秒）         :      0.230647
ベクトル命令実行時間（秒）  :      4.846089
全命令実行数               :    380150140.
ベクトル命令実行数         :    205713526.
ベクトル命令実行要素数     :  52468073111.
浮動小数点データ実行要素数  :  14758745137.
MOPS 値                    :  10718.595612
MFLOPS 値                  :   3005.043295
平均ベクトル長             :    255.054075 ①
ベクトル演算率（%）        :     99.668639 ②
メモリ使用量（MB）         :   1820.031250
MIPS 値                    :     77.402761
命令キャッシュミス（秒）    :      0.014333
オペランドキャッシュミス（秒）:    0.010467
バンクコンフリクト時間（秒）  :    0.244283 ③
```

図 4.4.2　Proginf の出力例

最も注目すべき項目は，①平均ベクトル長，②ベクトル演算率，③バンクコンフリクト時間である．

平均ベクトル長は，1 つのベクトル命令で演算を行った演算要素数の平均値である．SX-7 の場合，1 つのベクトル命令で最大 256 要素の演算を行うので，平均ベクトル長は 256 を超えることはないが, 256 に近い値であるほど効率の高い演算を行うことができる．

ベクトル演算率は，ベクトル命令を使用して演算が行われた割合を示し，

$$\text{ベクトル演算率} = \frac{(\text{ベクトル命令実行要素数}) \times 100}{(\text{全命令実行数} - \text{ベクトル命令実行数} + \text{ベクトル命令実行要素数})}$$

となる. 厳密にはベクトル化率とは言えないが, ほぼベクトル化率に近い値と考えて良い．したがってベクトル演算率が 100%に近づくほど高い性能を得ることができる．

バンクコンフリクト時間は，CPU がメモリアクセスを行う際にメモリバンクが競合して待ちになる時間である. 全体時間に対するバンクコンフリクト時間の割合が高い場合は, メモリバンクの競合が発生する原因を究明し回避する手立てが必要となる．

並列処理時の Proginf には，CPU が同時に実行した時間(Concurrent Time)が併せて出力される．負荷バランスが各 CPU に均一であるときは，この数値が図 4.4.3 のように揃っ

た値となる．

```
最大同時実行可能プロセッサ数          :              4.
  1 台以上で実行した時間（秒）        :       1.061972
  2 台以上で実行した時間（秒）        :       1.058297
  3 台以上で実行した時間（秒）        :       1.057302
  4 台以上で実行した時間（秒）        :       0.945636
```

図 4.4.3　実行時の Concurrent Time の例

（3）ftrace 情報（FLOW TRACE ANALYSIS LIST）

Proginf ではプログラム全体の性能情報しか得られないため，プログラムの最適化を検討するサブルーチン, ループを絞り込むことが難しい. SX-7 のコンパイラの機能として，サブルーチンや関数といった副プログラム単位の性能情報を採取する ftrace がある．コンパイルオプションに-ftrace を追加したロードモジュールを実行すると，標準エラー出力に ftrace 情報が出力される．図 4.4.4 に ftrace 情報の採取方法を示す.

```
%sxf90 -ftrace test.f
%./a.out
```

図 4.4.4　ftrace 情報の採取方法

ftrace 情報は，Proginf と異なり，副プログラムの入口と出口で計測ポイントを設定するため，プログラム実行時にオーバヘッドが加わる．特に副プログラムの呼び出し回数が非常に多い場合や, 副プログラムの呼び出しのネストが深い場合はプログラムの実行時間が長くなる場合がある．図 4.4.5 に ftrace 情報の出力例を示す.

```
①          ②          ③                ④                      ⑤    ⑥                          ⑦
PROG.UNIT  FREQUENCY  EXCLUSIVE        AVER.TIME    MOPS MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )     [msec]                 RATIO V.LEN     TIME   MISS    MISS      CONF

cccc             128    90.620( 65.5)   707.966 6282.9 4645.1 99.58 256.0    88.793  0.0002  0.0008  0.0001
bbbb             100    33.221( 24.0)   332.209 11836.7 7722.6 99.81 250.0   33.209  0.0004  0.0063  0.0000
dddd              15    14.092( 10.2)   939.465 6057.3 1142.9 99.06 256.0    14.092  0.0000  0.0000  0.0000
main               1     0.291(  0.2)   291.189 7574.0 3694.6 98.44 252.7     0.291  0.0002  0.0002  0.0000
aaaa               1     0.039(  0.0)    39.028 1890.4  164.6 96.06 125.2     0.039  0.0000  0.0000  0.0000
-------------------------------------------------------------------------------------------------------------
total            245   138.263(100.0)   564.338 7595.8 5024.3 99.62 253.7   136.424  0.0009  0.0072  0.0001
```

図 4.4.5　ftrace 情報の出力例

① PROG.UNIT…副プログラム (サブルーチンあるいは関数)名.
② FREQUENCY…副プログラムが実行された回数.
③ EXCLUSIVE TIME…副プログラムの実行された時間と全体時間からの割合.
④ AVER.TIME…副プログラムの一回当たりの実行時間.
⑤ V.OP RATIO…副プログラムのベクトル演算率.
⑥ AVER V.LEN…副プログラムの平均ベクトル長.
⑦ BANK CONF…副プログラムのバンクコンフリクト時間.

出力される順番は，副プログラムの実行時間の多いものからとなる.

ftrace 情報は並列化実行時にも採取することが可能であり，CPU ごとの情報を見ること

ができる．図 4.4.6 に，4CPU 利用して実行した場合の ftrace 情報の出力例を示す．図のとおり，並列化されたサブルーチンには，サブルーチン名の後ろに「$数字」が付く．CPU ごとの性能情報は，各サブルーチンの性能情報の下に，「-micro 数字」として表示される．

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]               RATIO V.LEN    TIME   MISS    MISS      CONF

main_$8         120   109.173( 80.6)   909.778 11724.6 5294.1 99.82 254.9  109.152  0.0015  0.0016  0.0000
 -micro1         30    27.290( 20.2)   909.677 11725.9 5294.7 99.82 254.9   27.288  0.0007  0.0008  0.0000
 -micro2         30    27.294( 20.2)   909.815 11724.1 5293.9 99.82 254.9   27.288  0.0003  0.0003  0.0000
 -micro3         30    27.294( 20.2)   909.809 11724.2 5294.0 99.82 254.9   27.288  0.0003  0.0003  0.0000
 -micro4         30    27.294( 20.2)   909.813 11724.2 5293.9 99.82 254.9   27.288  0.0003  0.0003  0.0000
  :
```

図 4.4.6　並列化実行時の ftrace 情報の出力例

また，副プログラム単位だけではなく，指定した範囲の性能情報を見ることも可能である．ftrace_region_begin，ftrace_region_end に挟まれた範囲の情報が出力される．図 4.4.7 に ftrace 情報範囲指定方法の例を示す．

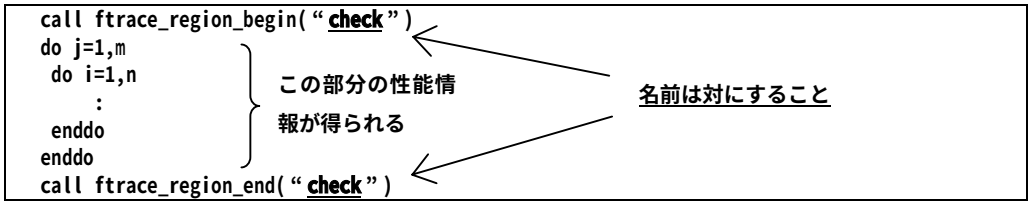

図 4.4.7　ftrace 情報範囲指定方法の例

これにより，do ループ単位にまで絞り込んで解析を行うことが可能になる．図 4.4.8 に ftrace 情報範囲指定の出力例を示す．

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]               RATIO V.LEN    TIME   MISS    MISS      CONF

func2           128    93.898( 66.0)   733.581 6063.5 4482.9 99.58 256.0   92.085  0.0017  0.0026  0.0001
bbbb            100    33.301( 23.4)   333.012 11808.2 7703.9 99.81 250.0   33.290  0.0002  0.0075  0.0000
func1            15    14.668( 10.3)   977.854 5819.5 1098.1 99.06 256.0   14.668  0.0011  0.0013  0.0001
main              1     0.296(  0.2)   296.430 7440.1 3629.3 98.44 252.7    0.296  0.0003  0.0002  0.0000
aaaa              1     0.043(  0.0)    43.081 1712.5  149.1 96.06 125.2    0.043  0.0000  0.0000  0.0000
-------------------------------------------------------------------------------------------------------------
total           245   142.207(100.0)   580.436 7385.1 4885.0 99.62 253.7  140.381  0.0034  0.0116  0.0002

check           100    33.119( 23.3)   331.189 11833.6 7746.2 99.82 250.0   33.108  0.0002  0.0073  0.0000
-------------------------------------------------------------------------------------------------------------
```

図 4.4.8　ftrace 情報範囲指定の出力例

４．５　ベクトル化チューニング事例

４．５．１　入出力文を含む do ループのベクトル化

SX-7 では，do ループの一部にベクトル化対象外の文があると，ベクトル化されない．以下に，do ループ内からベクトル化の阻害要因の一つである入出力文の除去を行い，ベクトル化を促進した例を示す．

（1）性能の分析

図 4.5.1.1 に Proginf を示す．図中①に示すように，ベクトル演算率が 0.7%と低いことがわかる．

```
      ******    プログラム 情報    ******
経過時間（秒）                :      1130.708891
ユーザ時間（秒）              :      1096.143670
システム時間（秒）            :         1.267607
ベクトル命令実行時間（秒）    :         0.256928
全命令実行数                  :    349195261811.
ベクトル命令実行数            :         9008178.
ベクトル命令実行要素数        :      2290285204.
浮動小数点データ実行要素数    :     81404252932.
MOPS 値                       :       320.648240
MFLOPS 値                     :        74.264218
平均ベクトル長                :       254.245110
ベクトル演算率（%）           :         0.651618   ①
メモリ使用量（MB）            :        64.031250
MIPS 値                       :       318.567056
命令キャッシュミス（秒）      :         0.101621
オペランドキャッシュミス（秒）:       128.705834
バンクコンフリクト時間（秒）  :         0.000129
```

図 4.5.1.1　Proginf

図 4.5.1.2 に ftrace 情報を示す．図中①に示すように，コストのほぼ 100.0%を占めるサブルーチン func のベクトル演算率が 0.0%であり，ほとんどベクトル化されていないことがわかる．

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]               RATIO V.LEN    TIME   MISS    MISS      CONF

func             100  1095.888(100.0) 10958.877  318.6   73.5  0.00  69.2    0.003  0.1007 128.7055  0.0001
                                                                  ①
main               1     0.254(  0.0)   254.169 9072.2 3435.5 99.22 255.0    0.254  0.0001  0.0001  0.0000
-------------------------------------------------------------------------------------------------------------
total            101  1096.142(100.0) 10852.890  320.6   74.3  0.65 254.2    0.257  0.1008 128.7055  0.0001
```

図 4.5.1.2　ftrace 情報

図 4.5.1.3 にサブルーチン func の編集リストを示す. 図に示すように do ループ内に 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，図中「+」記号が示すように，do ループ全体がベクトル化されていない.

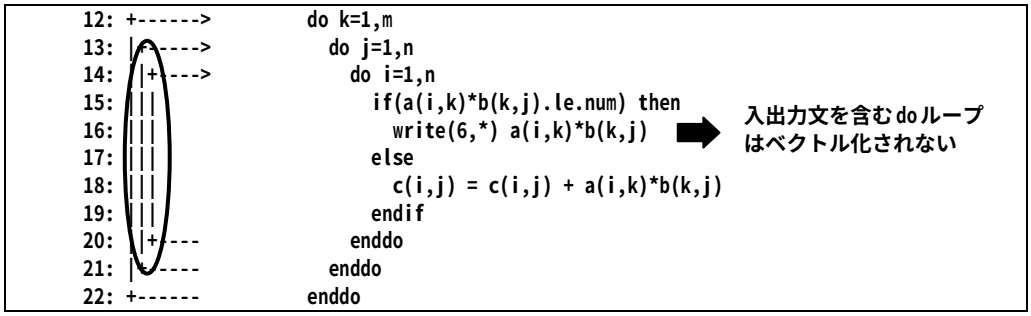

図 4.5.1.3　サブルーチン func の編集リスト

（2）入出力文の do ループ外への移動

do ループ内から，ベクトル化対象外の文である write 文を別の do ループに切り分けることにした.

図 4.5.1.4 にソース修正後の編集リストを示す．図に示すように，作業配列 wk を設け，write 文に渡すための値はその中に格納した．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り，図中「V」記号が示すように，ベクトル化されるようになった.

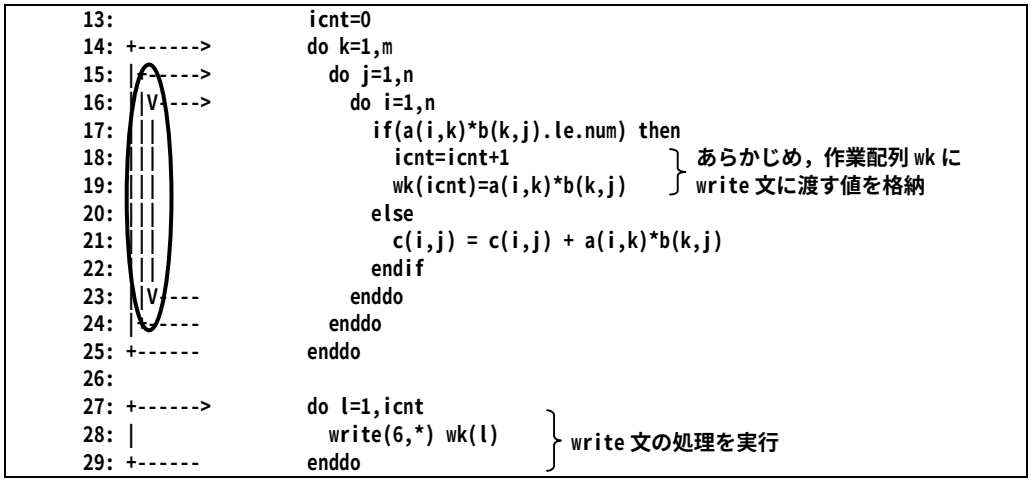

図 4.5.1.4　ソース修正後の編集リスト

（3）ソース修正による効果

図 4.5.1.5 にソース修正後の ftrace 情報を示す．図に示すように，サブルーチン func のベクトル演算率が 99.1%に向上し（修正前は 0.0%），プログラム全体のベクトル演算率は 99.1%に向上した（修正前は 0.7%）．また，実行時間は 48.4 秒に短縮された（修正前は 1096.1 秒）．

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]               RATIO V.LEN    TIME   MISS    MISS      CONF

func             100   48.137( 99.5)   481.373 4501.9 1673.0 99.10 256.0   43.210  0.0256  0.3238  0.0000
main               1    0.254(  0.5)   253.695 9089.1 3441.9 99.22 255.0    0.254  0.0001  0.0000  0.0000
-------------------------------------------------------------------------------------------------------------
total            101   48.391(100.0)   479.119 4526.0 1682.2 99.10 256.0   43.464  0.0257  0.3238  0.0000
```

図 4.5.1.5　ソース修正後の ftrace 情報

４．５．２　副プログラムを含む do ループのベクトル化

副プログラムの引用(CALL 文や関数呼び出し)を含む do ループはベクトル化できない．この場合，副プログラムをインライン展開することによりベクトル化が可能になる．インライン展開はコンパイラオプション-pi によって自動的に行われるが，自動インライン展開阻害要因がある場合，手動でソース修正を行う必要がある．以下に手動でインライン展開を行った例を示す．また，自動インライン展開阻害要因の詳細については 4.5.8 章(4)で紹介する．

（1） 性能の分析

図 4.5.2.1 に Proginf を示す．図中①に示すように，ベクトル演算率が 2.1%と低いことがわかる．

```
      ******    プログラム 情報    ******
経過時間（秒）                :      1188.872381
ユーザ時間（秒）              :      1183.763346
システム時間（秒）            :         2.394112
ベクトル命令実行時間（秒）    :         2.296196
全命令実行数                  :    230944773788.
ベクトル命令実行数            :        38487182.
ベクトル命令実行要素数        :      5048783549.
浮動小数点データ実行要素数    :      1259607979.
MOPS 値                       :       199.326218
MFLOPS 値                     :         1.064071
平均ベクトル長                :       131.180910
ベクトル演算率（%）           :         2.139722  ①
メモリ使用量（MB）            :      4848.031250
MIPS 値                       :       195.093702
命令キャッシュミス（秒）      :         0.993977
オペランドキャッシュミス（秒）:       149.308159
バンクコンフリクト時間（秒）  :         0.000010
```

図 4.5.2.1　Proginf

図 4.5.2.2 に ftrace 情報を示す．①のようにサブルーチン sub のベクトル演算率は 0.0%であり，②のようにサブルーチン main のベクトル演算率は 29.82%である．よって，ともにほとんどベクトル化されていないことがわかる．

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME    MOPS  MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec]( % )    [msec]                RATIO V.LEN    TIME   MISS    MISS      CONF
sub        209715200  125.325( 59.1)     0.001    47.5     5.0  0.00   0.0    0.000  0.0615 41.6679  0.0000
                                                                ①
main             1    86.776( 40.9) 86776.407   195.1     7.3 29.82 131.2    2.296  0.1341  2.0274  0.0000
                                                                ②
-------------------------------------------------------------------------------------------------------------
total      209715201  212.102(100.0)     0.001   107.9     5.9 22.06 131.2    2.296  0.1956 43.6953  0.0000
```

図 4.5.2.2　ftrace 情報

図 4.5.2.3 に編集リストを示す．図中の「+」記号が示すように，call 文を含む do ループはベクトル化されていない．また，呼び出される側のサブルーチン sub には do ループがないのでベクトル化されない．

```
 1:                  program main
…
 9:
10: +------>         do j=1,m
11: |+---->             do i=1,n
…   ||                    …
16: ||                    call sub(a(i,j),b(j,i),x(i,j))   } call 文を含む do ループ
…   ||                    …                                  はベクトル化されない
22: |+-----             enddo
23: +------           enddo
…
30:                  subroutine sub(a,b,x)
31:                          x = a*b
32:                         if(x.le.0.25) then
33:                           x = x + 0.25
34:                         else
35:                           x = x - 0.25
36:                         endif
37:                  end
…
```

図 4.5.2.3　編集リスト

（2）インライン展開

図 4.5.2.4 にソース修正後の編集リストを示す．call 文を含む do ループをベクトル化するために, サブルーチン sub をインライン展開した. インライン展開を行ったことにより, 図中の「V」記号が表すように do ループの内側がベクトル化されるようになった.

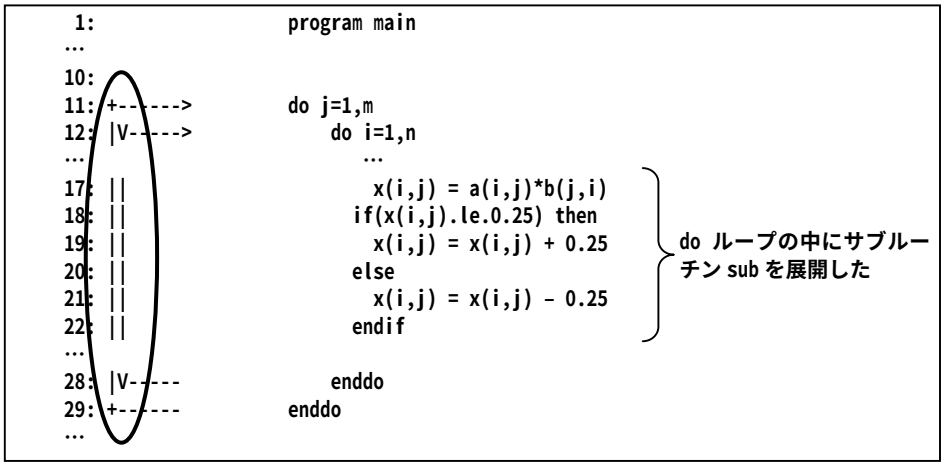

図 4.5.2.4　ソース修正後の編集リスト

（3）インライン展開の効果

図 4.5.2.5 にソース修正後の ftrace 情報を示す．インライン展開により sub は main に吸収されていることがわかる．

図中②に示すように，プログラム全体のベクトル演算率は 98.0%に向上した（修正前は 22.1%）．この結果，図中①のように全体の実行時間は 6.8 秒に短縮された（修正前は 212.1 秒）．

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONF
main	1	6.773(100.0)	6773.252	1108.1	216.9	98.00	154.9	6.773	0.0006	0.0005	3.6073
total	1	6.773(100.0)	6773.252	1108.1	216.9	98.00	154.9	6.773	0.0006	0.0005	3.6073

① ②

図 4.5.2.5　ソース修正後の ftrace 情報

４．５．３　重なりのあるリストベクトルのベクトル化

定義・参照を行っている配列の添え字が間接参照（リストベクトル）の場合，添え字の重なりの有無をコンパイラは判断することができず，ベクトル化を行うことができない．添え字に重なりがないことがわかっていればNODEP 指示行を挿入することでベクトル化できるようになる．NODEP は配列の左辺と右辺に依存関係がないことを指定する指示行である．

添え字に重なりがあるかどうかわからない場合，止まり木の手法を用いてソースを修正することでベクトル化できるようになる．

以下に，止まり木の手法を用いたベクトル化の例を示す．

（1） 性能の分析

図 4.5.3.1 に Proginf を示す．図中①に示すように，ベクトル演算率が 84.9%と低いことがわかる．

```
      ******    プログラム 情報    ******
   経過時間（秒）                :       270.278571
   ユーザ時間（秒）              :       250.919937
   システム時間（秒）            :         0.582652
   ベクトル命令実行時間（秒）    :        39.477710
   全命令実行数                  :     25106008471
   ベクトル命令実行数            :       787386595
   ベクトル命令実行要素数        :    136912368041
   浮動小数点データ実行要素数    :     52729143899
   MOPS 値                       :       990.953165
   MFLOPS 値                     :       324.082312
   平均ベクトル長                :       173.882015
   ベクトル演算率（%）           :        84.916906  ①
   メモリ使用量（MB）            :      3392.031250
   MIPS 値                       :       154.305810
   命令キャッシュミス（秒）      :         0.202416
   オペランドキャッシュミス（秒）:        43.815168
   バンクコンフリクト時間（秒）  :         1.574089
```

図 4.5.3.1　Proginf

図 4.5.3.2 に ftrace 情報を示す. 図中①に示すようにサブルーチン sub_a はコストの 53.0%を占めており，ベクトル演算率は②に示すように 0.11%である．よって，実行時間を短縮するために sub_a のベクトル演算率を上げる必要がある．

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]                RATIO V.LEN    TIME   MISS    MISS      CONF

sub_a           603   132.324( 53.0)    219.442 212.2   96.1  0.11  64.3    0.015  0.0375 27.1782  0.0000
                            ①                                    ②
sub_b         97660    60.360( 37.3)      0.618 265.8   91.4 61.20  39.0   16.913  0.0514 12.0682  0.0006

sub_c           200    32.607( 20.2)    163.037 299.8   28.2 22.38 209.6    1.738  0.0456  2.3782  0.0000
sub_d           401     9.372(  5.8)     23.372 6565.1 2060.8 99.80 255.9    9.369  0.0034  0.0016  1.2770
-------------------------------------------------------------------------------------------------------------
total        643808   249.871(100.0)      0.251  997.5  326.2 84.91 173.9   38.535  0.2192 41.7646  1.4999
```

図 4.5.3.2　ftrace 情報

図 4.5.3.3 に sub_a の編集リストを示す．図中の「S」記号より，配列 sum の演算はベクトル化されていないことがわかる．これは sum の添え字 i,j がリストベクトルになっているため，添え字の重なりの有無をコンパイラが判断することができないからである．

```
    7: V------>        do m=1,num
    8: |                 i=abc(m,1)        } 配列 sum の添え字
    9: |                 j=abc(m,2)        } として参照される
   10: |                 def1=xx(i+1,1)-yy(j-1,1)
   11: |                 def2=xx(i-1,2)+yy(j+1,2)
   12: |                 ghi1=ww(i-1,1)+zz(j+1,1)
   13: |                 ghi2=ww(i+1,1)-zz(j-1,1)
   14: |      S          sum(i,  j  )=sum(i,  j  )+pi*def1*ghi1/vv(i,j)
   15: |      S          sum(i+1,j  )=sum(i+1,j  )+pi*def1*ghi2/vv(i,j)   総和を行う
   16: |      S          sum(i,  j+1)=sum(i,  j+1)+pi*def2*ghi1/vv(i,j)   配列 sum の演算
   17: |      S          sum(i+1,j+1)=sum(i+1,j+1)+pi*def2*ghi2/vv(i,j)
   18: V------        enddo
```

図 4.5.3.3　sub_a の編集リスト

（2）止まり木の手法の適用方法

図 4.5.3.4 に止まり木の手法を用いて sub_a を修正した後の編集リストを示す. 止まり木の手法とは, 配列の次元を一つ拡張（図 4.5.3.4 中では配列 sum の添え字 k）した作業配列を用いることにより, 添え字の重なりをなくす方法である. はじめに作業配列を初期化し, 最後に作業配列の足しこみを行う. これにより添え字に重なりのある配列をベクトル化することができる．またこの際，拡張した次元の大きさを最大ベクトル長（SX-7 では 256）とすることにより，効率のよいベクトル化を行うことができる.

```
    30: ++V====          wksum=0.0d0          ← 作業配列を初期化
    31:
    32: +------>         do mm=1,num,256
    33: |                  nn=256
    34: |                  if(num-mm+1.le.256) nn=num-mm+1
    35: |V----->           do k=1,nn
    36: ||                   m=mm+k-1
    37: ||                   i=abc(m,1)
    38: ||                   j=abc(m,2)
    39: ||                   def1=xx(i+1,1)-yy(j-1,1)
    40: ||                   def2=xx(i-1,2)+yy(j+1,2)
    41: ||                   ghi1=ww(i-1,1)+zz(j+1,1)
    42: ||                   ghi2=ww(i+1,1)-zz(j-1,1)
    43: ||                   wksum(k,i,  j  )=wksum(k,i,  j  )+pi*def1*ghi1/vv(i,j)
    44: ||                   wksum(k,i+1,j  )=wksum(k,i+1,j  )+pi*def1*ghi2/vv(i,j)
    45: ||                   wksum(k,i,  j+1)=wksum(k,i,  j+1)+pi*def2*ghi1/vv(i,j)
    46: ||                   wksum(k,i+1,j+1)=wksum(k,i+1,j+1)+pi*def2*ghi2/vv(i,j)
    47: |V-----            enddo
    48: +------          enddo
    49:
    50: +------>         do j=1,jmax          ← 作業配列の足しこみ
    51: |+----->           do i=1,imax
    52: ||V---->             do k=1,256
    53: |||                    sum(i,j)=sum(i,j)+wksum(k,i,j)
    54: ||V----              enddo
    55: |+-----            enddo
    56: +------          enddo
```

（43〜46 行目：配列の次元を一つ拡張した（添え字 k））

図 4.5.3.4　ソース修正後の sub_a の編集リスト

（3） 止まり木の手法の効果

図 4.5.3.5 にソース修正後の ftrace 情報を示す．図中①に示すように sub_a の実行時間は 14.7 秒に短縮された（修正前は 132.3 秒）. これは, 図中②のようにベクトル演算率が 99.8% に向上した（修正前は 0.1%）ことおよび図中③のように平均ベクトル長が 255.6 に向上した（修正前は 64.3）ことによる.

sub_a の演算性能が向上した結果，全体の実行時間は図中④に示すように 132.3 秒に短縮された（修正前は 249.9 秒）.

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]                RATIO V.LEN    TIME   MISS     MISS      CONF

sub_a          97659    60.365( 45.6)     0.618  265.7   91.4 61.20  39.0   16.887  0.0545 12.1005  0.0006
sub_c            200    32.624( 24.7)   163.119  299.6   28.2 22.38 209.6    1.733  0.0608  2.3870  0.0000
sub_b            603    14.673( 11.1)    24.335 5350.5 1254.7 99.80 255.6   14.505  0.0135  0.0084  0.6498
                        ①                                     ②    ③
sub_d            401     9.378(  7.1)    23.385 6561.4 2059.6 99.80 255.9    9.374  0.0042  0.0022  1.2822
-------------------------------------------------------------------------------------------------------------
total         643807   132.303(100.0)     0.190 1453.3  445.9 91.55 183.3   52.999  0.2737 14.6271  2.1547
                       ④
```

図 4.5.3.5　ソース修正後の ftrace 情報

４．５．４　ベクトル長の拡大

平均ベクトル長が小さいと，図 4.2.3 に示すようにベクトル処理の効果が低く，実行時間が長くなる. 以下に, 平均ベクトル長を拡大し, ベクトル処理の効率を上げた例を示す.

（1） 性能の分析

図 4.5.4.1 に Proginf を示す．図中①に示すように，平均ベクトル長は 129.9 であり，十分に大きいとは言えない.

```
      ******    プログラム  情報    ******
   経過時間（秒）                :        337.754345
   ユーザ時間（秒）              :        329.712521
   システム時間（秒）            :          0.094685
   ベクトル命令実行時間（秒）    :        308.082955
   全命令実行数                  :      35179138465.
   ベクトル命令実行数            :      10413430205.
   ベクトル命令実行要素数        :    1352288967785.
   浮動小数点データ実行要素数    :     365195719423.
   MOPS 値                       :       4176.531331
   MFLOPS 値                     :       1107.618594
   平均ベクトル長                :        129.860088   ①
   ベクトル演算率（%）           :         98.201545
   メモリ使用量（MB）            :         48.031250
   MIPS 値                       :        106.696398
   命令キャッシュミス（秒）      :          0.005801
   オペランドキャッシュミス（秒）:          0.013908
   バンクコンフリクト時間（秒）  :          0.000002
```

図 4.5.4.1　Proginf

図 4.5.4.2 に ftrace 情報を示す. 図中①に示すように, サブルーチン solve はコストの 100.0% を占めており，②のようにその平均ベクトル長は 129.9 である.

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]              RATIO V.LEN    TIME   MISS    MISS      CONF
solve          2000   326.709(100.0)   163.354 4214.9 1117.8 98.20 129.9  306.012 0.0066  0.0107  0.0000
                                ①                                  ②
output            1     0.044(  0.0)    43.716  262.6    7.7  3.27  12.0    0.002 0.0002  0.0005  0.0000
main              1     0.002(  0.0)     1.773   56.5    0.0  0.00   0.0    0.000 0.0004  0.0004  0.0000
input             1     0.000(  0.0)     0.211  762.5   37.0 89.58  51.2    0.000 0.0000  0.0000  0.0000
-------------------------------------------------------------------------------------------------------
total          2003   326.755(100.0)   163.133 4214.3 1117.6 98.20 129.9  306.014 0.0073  0.0116  0.0000
```

図 4.5.4.2　ftrace 情報

図 4.5.4.3 にサブルーチン solve の編集リストを示す．図中の「V」記号が表すように，多重ループの最内ループがベクトル化の対象になっている．最内ループのループ長は nmax-1=30 であるにも関わらず，solve の平均ベクトル長が 30 ではなく図 4.5.4.2 の②に示すように 129.9 になっているのは，最内ループ内の演算をコンパイラが総和型のマクロ演算に置き換えているためである．マクロ演算への置き換えは，図 4.5.4.3 の上部に示したコンパイルメッセージよりわかる．

マクロ演算への置き換えにより平均ベクトル長が大きくなってはいるが，129.9 ではまだ十分な大きさではなく，演算性能を上げるためにはベクトル長の拡大が必要となる．

```
f90: vec(1): sample554.f, line 49: ループ全体をベクトル化する
f90: vec(26): sample554.f, line 50: Sum/InnerProd のマクロ演算としてベクトル化を行う
f90: vec(26): sample554.f, line 52: Sum/InnerProd のマクロ演算としてベクトル化を行う
f90: vec(26): sample554.f, line 54: Sum/InnerProd のマクロ演算としてベクトル化を行う
f90: vec(26): sample554.f, line 56: Sum/InnerProd のマクロ演算としてベクトル化を行う


    3:                  parameter(mmax=63,nmax=31)
   …
   46: +------>         do m=1,mmax
   47: |+----->          do k=1,nmax
   48: ||+---->           do j=1,mmax-1
   49: |||V--->            do i=1,nmax-1
   50: |||                  ph1(k,m)=ph1(k,m)
   51: |||                +           +chg1(k,m)*engy(i,  j  )+chg2(k,m)*dens(i,  j  )
   52: |||                  ph2(k,m)=ph2(k,m)
   53: |||                +           +chg1(k,m)*engy(i+1,j  )+chg2(k,m)*dens(i+1,j  )
   54: |||                  ph3(k,m)=ph3(k,m)
   55: |||                +           +chg1(k,m)*engy(i,  j+1)+chg2(k,m)*dens(i,  j+1)
   56: |||                  ph4(k,m)=ph4(k,m)
   57: |||                +           +chg1(k,m)*engy(i+1,j+1)+chg2(k,m)*dens(i+1,j+1)
   58: |||V--              enddo
   59: ||+----             enddo
   60: |+-----            enddo
   61: +------           enddo
```

図 4.5.4.3　サブルーチン solve の編集リスト

（2）ベクトル長の拡大方法

図 4.5.4.4 にソース修正後の編集リストを示す．図 4.5.4.3 中の外側 2 つのループを融合し，さらに，融合したループを内側ループと入れ替えることでベクトル化の対象としている．ループを融合することができるのは，ループ内にある配列 ph1,ph2,ph3,ph4 の添え字 k が 1 から nmax まで，添え字 m は 1 から mmax まで，いずれも連続して参照されているため，k*m を 1 つの変数 km として扱うことができるためである．

```
   46: +------>          do j=1,mmax-1
   47: |+----->           do i=1,nmax-1
   48: ||V---->            do km=1,nmax*mmax
   49: |||                   ph1(km,1)=ph1(km,1)
   50: |||                 +           +chg1(km,1)*engy(i,  j  )+chg2(km,1)*dens(i,  j  )
   51: |||                   ph2(km,1)=ph2(km,1)
   52: |||                 +           +chg1(km,1)*engy(i+1,j  )+chg2(km,1)*dens(i+1,j  )
   53: |||                   ph3(km,1)=ph3(km,1)
   54: |||                 +           +chg1(km,1)*engy(i,  j+1)+chg2(km,1)*dens(i,  j+1)
   55: |||                   ph4(km,1)=ph4(km,1)
   56: |||                 +           +chg1(km,1)*engy(i+1,j+1)+chg2(km,1)*dens(i+1,j+1)
   57: ||V----             enddo
   58: |+-----            enddo
   59: +------           enddo
```

図 4.5.4.4　ソース修正後の編集リスト

（3）ループ融合およびループ入れ換えの効果

図 4.5.4.5 にソース修正後の ftrace 情報を示す．図中②に示すようにサブルーチン solve の平均ベクトル長は 244.1 に改善された(修正前は 129.9)．これは，コンパイラによる総和型のマクロ演算への置き換えよりもループ融合の方がベクトル長拡大の効果が大きかったことを表している．これにより，図中①に示すように，実行時間は 21.8 秒に短縮された（修正前は 326.7 秒）．

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONF
solve	2000	21.798(99.8) ①	10.899	8689.7	5332.8	99.73	244.1 ②	21.795	0.0021	0.0029	0.0002
output	1	0.045(0.2)	44.785	256.4	7.5	3.27	12.0	0.005	0.0003	0.0005	0.0000
main	1	0.002(0.0)	1.782	56.2	0.0	0.00	0.0	0.000	0.0005	0.0003	0.0000
input	1	0.000(0.0)	0.263	611.2	29.7	89.58	51.2	0.000	0.0000	0.0000	0.0000
total	2003	21.844(100.0)	10.906	8671.6	5321.4	99.72	244.1	21.800	0.0029	0.0038	0.0002

図 4.5.4.5　ソース修正後の ftrace 情報

４．５．５　バンクコンフリクトの回避

バンクコンフリクトは SX-7 の演算性能を低下させる要因である．これは，配列データのアクセス方法によって発生する場合がある．以下に，配列の使い方の変更によってバンクコンフリクトを回避した例を示す．

（1）性能の分析

図 4.5.5.1 に Proginf を示す．図中①に示す平均ベクトル長，および②に示すベクトル演算率はともに高い値を示している．しかし，③に示すバンクコンフリクト時間は 1124.3 秒と，ユーザ時間の約 3 分の 1 を占めている．

```
      ******    プログラム 情報    ******
    経過時間（秒）                :      3651.261524
    ユーザ時間（秒）              :      3644.885959
    システム時間（秒）            :         3.021369
    ベクトル命令実行時間（秒）    :      3625.232560
    全命令実行数                  :    305051263468.
    ベクトル命令実行数            :     93131375684.
    ベクトル命令実行要素数        :  23742486609381.
    浮動小数点データ実行要素数    :   9489525377288.
    MOPS 値                       :      6572.059254
    MFLOPS 値                     :      2603.517773
    平均ベクトル長                :       254.935423  ①
    ベクトル演算率（%）           :        99.115320  ②
    メモリ使用量（MB）            :     37248.031250
    MIPS 値                       :        83.692951
    命令キャッシュミス（秒）      :         0.582369
    オペランドキャッシュミス（秒）:         0.226319
    バンクコンフリクト時間（秒）  :      1124.310413  ③
```

図 4.5.5.1　Proginf

図 4.5.5.2 に ftrace 情報を示す．図中①に示すようにサブルーチン prog_do のバンクコンフリクト時間は 1124.1 秒である．これは全バンクコンフリクト時間のほとんどを占めていることがわかる．

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]               RATIO V.LEN    TIME   MISS    MISS      CONF

prog_do         4495  3629.111( 99.6)   807.366 6595.6 2613.7 99.13 255.1 3621.125  0.1110  0.1305 1124.1265
                                                                                                        ①
prog_sim           1     7.927(  0.2)  7926.550  227.0    0.7  4.41   1.2    1.518  0.0055  0.0540    0.0001
prog_read          1     4.999(  0.1)  4999.002  248.2    1.3  0.03 231.3    0.000  0.4113  0.0236    0.0000
  …
-----------------------------------------------------------------------------------------------------------------
total           5945  3644.825(100.0)   613.091 6572.2 2603.6 99.12 254.9 3625.232  0.5561  0.2160 1124.3104
```

図 4.5.5.2　ftrace 情報

図 4.5.5.3 にサブルーチン prog_do の編集リストを示す．図中に示すように二重 do ループの内側 do ループがベクトル化されている．内側 do ループ内で，配列 para_g(i,j)は，内側 do ループの変数 j が配列要素の二次元目となっている．よって，内側 do ループのアクセスは，i のサイズ NUM(=98700)要素飛びであることがわかる．これがバンクコンフリクト発生の原因となっている．

```
    2:                  PARAMETER ( NUM = 98700 )
    3:                  real para_g( NUM, NUM )
  ...
  252: |V----->            do j = 1, ncells
  253: ||                    para_g(i,j)=wbuf(j)
  254: |V-----             end do
  ...
 1030: +------>          do 6100 i = 1, ncells
 1031: |                    para_f = F0
 1032: |V----->            do 6110 j = 1, ncells
 1033: ||                     para_f = para_f + para_g( i, j ) * wary(j)
 1034: |V-----     6110    end do
  ...
 1054: +------     6100 end do
```

図 4.5.5.3　サブルーチン prog_do の編集リスト

（2）バンクコンフリクト回避の方法

配列が一次元目からアクセスされるように配列の使い方を変更し, 配列へのアクセスが連続となるようにする.

図 4.5.5.4 にソース修正後のサブルーチン prog_do の編集リストを示す. 図に示すように, 配列 para_g にデータを格納する際，一次元目からデータを格納するようにした．これにより，para_g(i,j)を para_g(j,i)と置き換えて，内側 do ループで一次元目の要素にアクセスするようにした.

```
    2:                  PARAMETER ( NUM = 98700 )
    3:                  real para_g( NUM, NUM )
  ...
  252: |V----->            do j = 1, ncells
  253: ||                    para_g(j,i)=wbuf(j)
  254: |V-----             end do
  ...
  ...
 1030: +------>          do 6100 i = 1, ncells
 1031: |                    para_f = F0
 1032: |V----->            do 6110 j = 1, ncells
 1033: ||                     para_f = para_f + para_g( j, i ) * wary(j)
 1034: |V-----     6110    end do
  ...
 1054: +------     6100 end do
```

para_g に，一次元目からデータを格納する

図 4.5.5.4　ソース修正後のサブルーチン prog_do の編集リスト

（3）ソース修正による効果

図 4.5.5.5 にソース修正後の ftrace 情報を示す．図中①に示すように，サブルーチン prog_do のバンクコンフリクト時間が 0.1 秒に短縮された（修正前は 1124.1 秒）．これに伴い，実行時間は 2476.2 秒に短縮された（修正前は 3644.8 秒）．

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONF
prog_do	4495	2460.317(99.4)	547.345	9425.9	3735.3	99.13	255.1	2452.804	0.4070	0.4658	0.0622 ①
prog_sim	1	8.110(0.3)	8109.692	221.8	0.7	4.41	1.2	1.595	0.0077	0.0682	0.0001
prog_read	1	5.203(0.2)	5202.641	238.5	1.2	0.03	231.3	0.000	0.4970	0.0419	0.0000
…											
total	5945	2476.232(100.0)	416.523	9372.6	3712.9	99.11	254.9	2456.819	0.9397	0.5902	0.0647

図 4.5.5.5　ソース修正後の ftrace 情報

４．５．６　行列積ループのライブラリ置き換え

do ループ中に行列積の演算が含まれている場合，SX のコンパイラはより高速な演算を行うためにライブラリへの置き換えを行う．以下に，do ループ内の演算を行列積の形に修正した例を示す．この修正によりコンパイラがライブラリへの置き換えを行うことが可能になった．

（1） 性能の分析

図 4.5.6.1 に Proginf を示す．図中に示す①平均ベクトル長は 248.2，②ベクトル演算率は 98.0%，③バンクコンフリクト時間は 0.0 秒と，特に問題はない．

```
      ******    プログラム 情報    ******
 経過時間（秒）                :       161.639781
 ユーザ時間（秒）              :       159.905034
 システム時間（秒）            :         0.487791
 ベクトル命令実行時間（秒）    :       121.182032
 全命令実行数                  :     20336217809.
 ベクトル命令実行数            :      3400523371.
 ベクトル命令実行要素数        :    844017228578.
 浮動小数点データ実行要素数    :    513889260236.
 MOPS 値                       :      5384.151476
 MFLOPS 値                     :      3213.715344
 平均ベクトル長                :       248.202155 ①
 ベクトル演算率（%）           :        98.032913 ②
 メモリ使用量（MB）            :       192.031250
 MIPS 値                       :       127.176846
 命令キャッシュミス（秒）      :         0.464495
 オペランドキャッシュミス（秒）:         3.940623
 バンクコンフリクト時間（秒）  :         0.000556 ③
```

図 4.5.6.1　Proginf

図 4.5.6.2 に ftrace 情報を示す．サブルーチン sub2 と sub3 では，②，⑤のベクトル演算率および③，⑥の平均ベクトル長にさほど差はない．しかし，図中①，④の MFLOPS 値を見ると，sub2 は sub3 に比べてはるかに演算効率が悪いことがわかる．

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONF
sub2	100	73.284(51.7)	732.839	6254.7	3507.7 ①	98.43 ②	250.0 ③	73.235	0.0173	0.0431	0.0000
sub4	1	35.266(24.9)	35265.901	263.7	7.9	3.00	10.9	1.980	0.0731	0.4528	0.0000
sub3	100	33.255(23.4)	332.547	11824.3	7714.6 ④	99.81 ⑤	250.0 ⑥	33.237	0.0127	0.0167	0.0000
sub1	1	0.035(0.0)	34.935	2111.9	183.8	96.06	125.2	0.035	0.0000	0.0000	0.0000
main	1	0.000(0.0)	0.419	28.2	0.0	0.00	0.0	0.000	0.0002	0.0001	0.0000
total	203	141.840(100.0)	698.718	6069.9	3623.0	98.03	248.2	108.487	0.1033	0.5127	0.0000

図 4.5.6.2　ftrace 情報

図 4.5.6.3 にサブルーチン sub2 の編集リスト，図 4.5.6.4 にサブルーチン sub3 の編集リストを示す．どちらも多重ループの処理であり，図中の「V」記号が表すようにベクトル化されている．さらに，図 4.5.6.4 中の矢印に示すコンパイルメッセージから，sub3 のループ中の処理が行列積のライブラリに置き換えられていることがわかる．これは sub3 の tflow(i,l,k)*eigen(j,l)の部分が行列積のパターンになっているため，コンパイラがライブラリへの置き換えを行ったことを表している．これに対して，sub2 の eigen(j,l)*vw(l)*pfk(i,l,k)の部分は行列積のパターンになっておらず，コンパイラによる置き換えは行われていない．

```
 47: +------>          do k=1,nz
 48: |+----->            do j=1,ny
 49: ||+---->              do l=1,lmax
 50: |||V--->                do i=1,nx
 51: |||                       tflow(i,j,k)=tflow(i,j,k)
 52: |||             +              +eigen(j,l)*vw(l)*pfk(i,l,k)
 53: ||V---                    enddo
 54: ||+----                  enddo
 55: |+-----                 enddo
 56: +------                enddo
```

図 4.5.6.3　サブルーチン sub2 の編集リスト

```
80  opt  (1800): 行列積ループをライブラリ呼び出しに置換した.  ⬅
      :
73: +------>          do k=1,nz
74: |*----->            do j=1,ny
75: ||V---->              do l=1,lmax
76: ||*---->                do i=1,nx
77: |||                       pflow(i,j,k)=pflow(i,j,k)
78: |||             +              +tflow(i,l,k)*eigen(j,l)
79: ||*---                    enddo
80: |V----                   enddo
81: |*-----                 enddo
82: +------                enddo
```

図 4.5.6.4　サブルーチン sub3 の編集リスト

（2）行列積のライブラリへ置き換え方法

サブルーチン sub2 のループ内の処理も行列積のパターンになるようにソースを修正した．図 4.5.6.5 にソース修正後の編集リストを示す．eigen(j,l)*vw(l)の演算を予め別のループで作業配列 wk(j,l)に置き換え，実際の演算部分では，wk(j,l)*pfk(i,l,k) となるようにした．これにより行列積のパターンとなり，コンパイラはライブラリへの置き換えを行っている．

```
60  opt  (1800): 行列積ループをライブラリ呼び出しに置換した.
      :
48: +------>          do l=1,lmax
49: |V----->            do j=1,ny                          予め eigen(j,l)*vw(l)
50: ||                    wk(j,l)=eigen(j,l)*vw(l) <-------の演算結果を wk(j,l)に
51: |V-----             enddo                              格納しておく
52: +------           enddo
53: +------>          do k=1,nz
54: |*----->            do j=1,ny
55: ||V---->              do l=1,lmax
56: |||*--->                do i=1,nx
57: ||||                    tflow(i,j,k)=tflow(i,j,k)          wk(j,l)*pfk(i,l,j)という
58: ||||             +                 +wk(j,l)*pfk(i,l,k) <---行列積のパターンになって
59: |||*---                enddo                              いる
60: ||V----               enddo
61: |*-----             enddo
62: +------            enddo
```

図 4.5.6.5　ソース修正後の編集リスト

（3）ソース修正による効果

図 4.5.6.6 にソース修正後の ftrace 情報を示す．図中②に示すように sub2 の MFLOPS 値は 7711.6 に向上した（修正前は 3507.7）．また，①に示すように sub2 の実行時間は 33.3 秒に短縮された（修正前は 73.3 秒）．これにより④に示すように全体の実行時間は 101.6 秒に短縮された（修正前は 141.8 秒）．

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONF
sub4	1	34.984(34.4)	34984.389	265.8	7.9	3.00	10.9	1.868	0.0765	0.6183	0.0000
sub3	100	33.280(32.8)	332.796	11815.5	7708.8	99.81	250.0	33.268	0.0015	0.0071	0.0001
sub2	100	33.268(32.8)	332.680	11820.0	7711.6	99.81	250.0	33.256	0.0014	0.0072	0.0000
		①			②						
sub1	1	0.034(0.0)	34.242	2154.6	187.6	96.06	125.2	0.034	0.0000	0.0000	0.0000
main	1	0.000(0.0)	0.358	33.0	0.0	0.00	0.0	0.000	0.0002	0.0001	0.0000
total	203	101.567(100.0)	500.328	7835.4	5054.6	98.68	248.1	68.427	0.0796	0.6327	0.0001
		③									

図 4.5.6.6　ソース修正後の ftrace 情報

４．５．７　重複演算の削除

ベクトル演算率が高く, ベクトル長が長くても, 高速化の余地が残っている場合がある. その一つとして, 一度の計算でよいにもかかわらず, 重複して計算されている場合がある. この場合，重複演算を削除することで，演算量の削減を行うことができる.

（１）性能の分析

図 4.5.7.1 に Proginf を示す．図中①に示す平均ベクトル長，および②に示すベクトル演算率はともに高い値を示している.

```
      ******    プログラム 情報    ******
経過時間（秒）                :       180.034357
ユーザ時間（秒）              :       112.506836
システム時間（秒）            :         0.234196
ベクトル命令実行時間（秒）    :       109.624226
全命令実行数                  :      6414937517.
ベクトル命令実行数            :      2957973850.
ベクトル命令実行要素数        :    757212925864.
浮動小数点データ実行要素数    :    411205216914.
MOPS 値                       :      6761.099289
MFLOPS 値                     :      3654.935391
平均ベクトル長                :       255.990406  ①
ベクトル演算率（%）           :        99.545537  ②
メモリ使用量（MB）            :      8256.031250
MIPS 値                       :        57.018202
命令キャッシュミス（秒）      :         0.008210
オペランドキャッシュミス（秒）:         0.950829
バンクコンフリクト時間（秒）  :         0.189765
```

図 4.5.7.1　Proginf

図 4.5.7.2 に ftrace 情報を示す．図中に示すとおり，コストが一番高いサブルーチンは，func2 である.

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE  BANK
                      TIME[sec](  % )    [msec]              RATIO V.LEN    TIME   MISS    MISS    CONF

func2            103   98.165( 87.3)   953.059 6856.9 4013.9 99.61 256.0   95.285 0.0066  0.9498  0.1193
func1             15   14.049( 12.5)   936.569 6076.0 1146.5 99.06 256.0   14.048 0.0004  0.0005  0.0705
main               1    0.291(  0.3)   290.931 7580.7 3697.9 98.44 252.7    0.291 0.0001  0.0001  0.0000
-------------------------------------------------------------------------------------------------------------
total            119  112.505(100.0)   945.416 6761.2 3655.0 99.55 256.0  109.624 0.0072  0.9504  0.1898
```

図 4.5.7.2　ftrace 情報

図 4.5.7.3 にサブルーチン func2 の編集リストを示す．図中の計算ｙ(i)*i-z(i+1)は, 最内側 do ループの変数 i で求めることができる．しかし外側 do ループによって m*m 回分繰り返し計算されている．これは，y(i)*i-z(i)の計算を m*m 回重複して行っていることになる.

```
    53:                  parameter(m=1024)
    54:                  dimension a(m,m),b(m,m),y(m),z(m+1),x(m)
    55: +------>         do k=1,m
    56: |+----->           do j=1,m
    57: ||V---->             do i=1,m
    58: |||                     x(k) = x(k) + a(i,k)*b(k,j) * ( y(i) * i - z(i+1) )
    59: ||V----              enddo
    60: |+-----            enddo
         ...
    64: +------          enddo
```

図 4.5.7.3　サブルーチン func2 の編集リスト

（2）重複演算削除の方法

図 4.5.7.4 にソース修正後のサブルーチン func2 の編集リストを示す. 図中に示すように, 三重 do ループの前にあらかじめ y(i)*i - z(i)を計算して作業配列 wrk に格納しておく．そして本体 do ループでは，上記作業配列 wrk を引用するようにした.

```
    53:                  parameter(m=1024)
    54:                  dimension a(m,m),b(m,m),y(m),z(m+1),x(m),wrk(m)
    55: V------>         do i=1,m                          } あらかじめ，計算を行い，
    56: |                    wrk(i) =  y(i) * i - z(i+1)   } 作業配列 wrk に格納
    57: V------          enddo
    58: +------>         do k=1,m
    59: |
    60: |+----->           do j=1,m
    61: ||V---->             do i=1,m
    62: |||                     x(k) = x(k) + a(i,k)*b(k,j) * wrk(i)
    63: ||V----              enddo
   ...
    68: +------          enddo
```

図 4.5.7.4　ソース修正後のサブルーチン func2 の編集リスト

（3）ソース修正による効果

図 4.5.7.5 にソース修正後の ftrace 情報を示す. 図に示すように, サブルーチン func2 の実行時間が 74.0 秒に短縮された（修正前は 98.1 秒）．またプログラム全体の実行時間が 88.1 秒に短縮された（修正前は 112.5 秒）.

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONF
func2	103	74.013(83.9)	718.575	6190.1	4576.6	99.58	256.0	72.520	0.0080	0.0378	0.0470
func1	15	13.883(15.7)	925.534	6148.5	1160.1	99.06	256.0	13.882	0.0031	0.0033	0.0000
main	1	0.292(0.3)	291.734	7559.8	3687.8	98.44	252.7	0.291	0.0002	0.0001	0.0000
total	119	88.188(100.0)	741.075	6188.1	4035.8	99.49	256.0	86.694	0.0113	0.0412	0.0471

図 4.5.7.5　ソース修正後の ftrace 情報

４．５．８　インライン展開

一回当たりの実行時間が短く，呼び出し回数の多い副プログラムは，呼び出し時のオーバヘッドが大きくなるので高速化を妨げる要因となる．以下に，副プログラムの自動インライン展開によるオーバヘッドの削減を行った例を示す．

（１）性能の分析

図 4.5.8.1 に ftrace 情報を示す．図中①に示すように，サブルーチン check は呼び出し回数が 111,765,600 回と多く，②のように一回当たりの実行時間が 0.001msec と短い．一般に，実行時間が短く，呼び出される回数が多い副プログラムは，インライン展開することで呼び出しのためのオーバヘッドが削減され，性能の向上が得られる．図中③，④のサブルーチン volume，param についても同様である．

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONF
solver	234650	152.913(25.6)	0.652	603.0	169.9	85.15	86.7	50.673	0.1158	41.9971	0.0131
calca	3610	142.035(23.8)	39.345	95.1	19.9	1.01	255.4	0.028	0.4892	50.2056	0.0002
calcb	3610	124.860(20.9)	34.587	105.3	22.2	1.04	255.4	0.029	0.7522	38.0813	0.0002
check	111765600 ①	56.564(9.5)	0.001 ②	75.1	0.0	0.00	0.0	0.000	0.0254	0.6623	0.0001
calcc	3610	51.250(8.6)	14.197	234.3	28.4	62.14	149.3	4.012	0.1474	22.7819	0.0000
volume ③	18627600	40.866(6.8)	0.002	154.8	0.4	0.00	0.0	0.000	0.2581	9.5769	0.0001
param ④	28013600	12.500(2.1)	0.000	56.0	0.0	0.00	0.0	0.000	0.0139	0.0147	0.0000
flawb	1173250	4.330(0.7)	0.004	101.0	24.6	0.00	0.0	0.000	0.3445	0.9191	0.0000
:											
total	161316393	597.195(100.0)	0.004	252.7	56.0	61.75	88.7	58.595	2.6638	166.1046	0.0149

図 4.5.8.1　ftrace 情報

（２）インライン展開の指定

SX-7 の FORTRAN コンパイラは，-pi オプションを指定することで自動インライン展開を行うことができる．以下，代表的なサブオプションについて説明する．

-pi　自動インライン展開を行うことを指定(既定値ではないので指定が必要)

line=α：　自動インライン展開の対象となる副プログラムの最大行数を指定（既定値は α=50 であるので，51 行以上の副プログラムを展開する場合は指定が必要）

nest=β：　自動インライン展開の対象となる副プログラムのネストの深さを指定（既定値は β=1）

expin=ファイル名/ディレクトリ名：

展開したい副プログラムと，展開される場所を記述したソースコードが別のファイルに記述されている場合，自動インライン展開の対象とならないので，明示的にどのファイルに記述されているのかを指定する．

exp=副プログラム名：

指定された副プログラムがインライン展開の対象となることを指定する．

（3）インライン展開の効果

図 4.5.8.2 に自動インライン展開オプションを追加した場合の ftrace 情報を示す. 図のようにサブルーチン check，volume，param がインライン展開され，①に示すように全体の実行時間が 322.5 秒に向上した(オプション指定前は 592.7 秒).

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]              RATIO V.LEN    TIME   MISS    MISS      CONF

calcc          3610   124.411( 38.2)    34.463  648.6  167.5 86.15  89.2   44.137 0.1938 31.4907  0.0234
calca          3610    93.255( 28.6)    25.832  245.0   61.8 41.19  87.0    5.907 0.3027 34.8644  0.0002
calcb          3610    92.126( 28.3)    25.520  242.6   61.6 41.68  86.1    5.882 0.2967 34.6228  0.0132
flawa       1173250     4.145(  1.3)     0.004  105.5   25.7  0.00   0.0    0.000 0.0495  1.0749  0.0000
flawb       1173250     3.844(  1.2)     0.003  112.9   27.3  0.00   0.0    0.000 0.0456  0.8623  0.0000
  :
----------------------------------------------------------------------------------------------------
total       2611845   325.703(100.0)     0.125  413.4  100.3 70.69  87.4   59.789 1.3123 103.9001  0.0379
                      ①
```

図 4.5.8.2　オプション追加時の ftrace 情報

（4）自動インライン展開の阻害要因

以下に挙げるような場合は，コンパイラは自動インライン展開ができない．インライン展開を阻害する要因を取り除くか，手動で展開する必要がある．

- 展開するルーチンが同じソース上で見つからない(-pi expin オプションを用いる)
- 展開ルーチンに構文エラーがある(構文エラーを修正する)
- 呼び出し手順で使用される引数が展開ルーチンの引数に一致しない(引数の個数と型を一致させる)
- 呼び出しルーチンと展開ルーチンの共通ブロックに衝突がある
- 展開するルーチンに NAMELIST が含まれる
- 展開するルーチンに SAVE 属性を持つ変数が含まれる
- 関数が展開されていて，それが DO WHILE 文または ELSE IF 文から呼び出される
- 展開ルーチンで引用される関数名が，呼び出し元ルーチンで使用される非関数名と衝突する
- 展開するルーチンに並列化制御オプション，強制並列化指示オプションまたは OpenMP 指示行が指定されている

４．５．９　乱数生成ルーチン

数値シミュレーションの多くで乱数を用いており，この際，計算機の演算によって生成される疑似乱数を使用することが多い．SX-7 上ではいくつかの乱数生成ルーチンを提供している．乱数を生成する場合，乱数生成に要する実行時間だけでなく，生成される乱数の性質等にも注意を払う必要がある．

（１）提供される乱数生成ルーチン

一度の呼び出しで 1 つの乱数を生成するものもあるが, 大規模シミュレーション等で使用する乱数を生成する場合は, 一度に複数の乱数を生成する一様乱数を使用することになる．以下，SX-7 で提供される一様乱数生成ルーチンである．

表 4.5.9.1　SX-7 で提供される一様乱数生成ルーチン

ルーチン名	乱数の周期	乱数の種類
random_number	2^{127}-1	
DJUFSP(ASL/SX)	2^{31}	線形合同法
DJUFLP(ASL/SX)	2^{250}-1	M 系列法
DJUFLR(ASL/SX)	可変	M 系列法

（２）性能値

乱数生成に要する時間について，100,000 個の一様乱数の生成を 5,000 回行うというベンチマークにて比較した結果が, 図 4.5.9.1 である. なお, ①に示すDJUFLRの周期は 2^{1396}-1 である.

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS  V.OP  AVER.   VECTOR I-CACHE O-CACHE   BANK
                      TIME[sec](  % )   [msec]                RATIO V.LEN    TIME   MISS    MISS      CONF

random_number   5000     2.533(  0.7)     0.507 2235.7  197.4 97.26 126.1    2.529  0.0011  0.0009  0.0000
DJUFSP          5000     0.571(  0.2)     0.114 9735.5  875.6 98.94 255.8    0.564  0.0020  0.0018  0.0000
DJUFLP          5000     1.955(  0.6)     0.391 1949.7  255.7 99.52 126.5    1.951  0.0001  0.0020  0.0000
DJUFLR ①        5000     0.889(  0.3)     0.178 4699.4  562.6 95.86 233.1    0.622  0.0003  0.0534  0.0000
```

図 4.5.9.1　一様乱数の性能値

この結果では，DJUFSPルーチンが最も高い性能であるが，周期が 2^{31}と短周期であるため，長周期の乱数を必要とするモンテカルロ法を用いた大規模なシミュレーションでは，DJUFLR ルーチンの利用をすすめる．

（3）乱数の検定

図 4.5.9.2 に 4 種類の乱数生成ルーチンについての検定結果を示す．100,000×5,000 個の乱数を生成し, 0.0 から 1.0 までを 20 分割した各領域への分布を調べたものである. 各ルーチンとも乱数値が分散されていることがわかる.

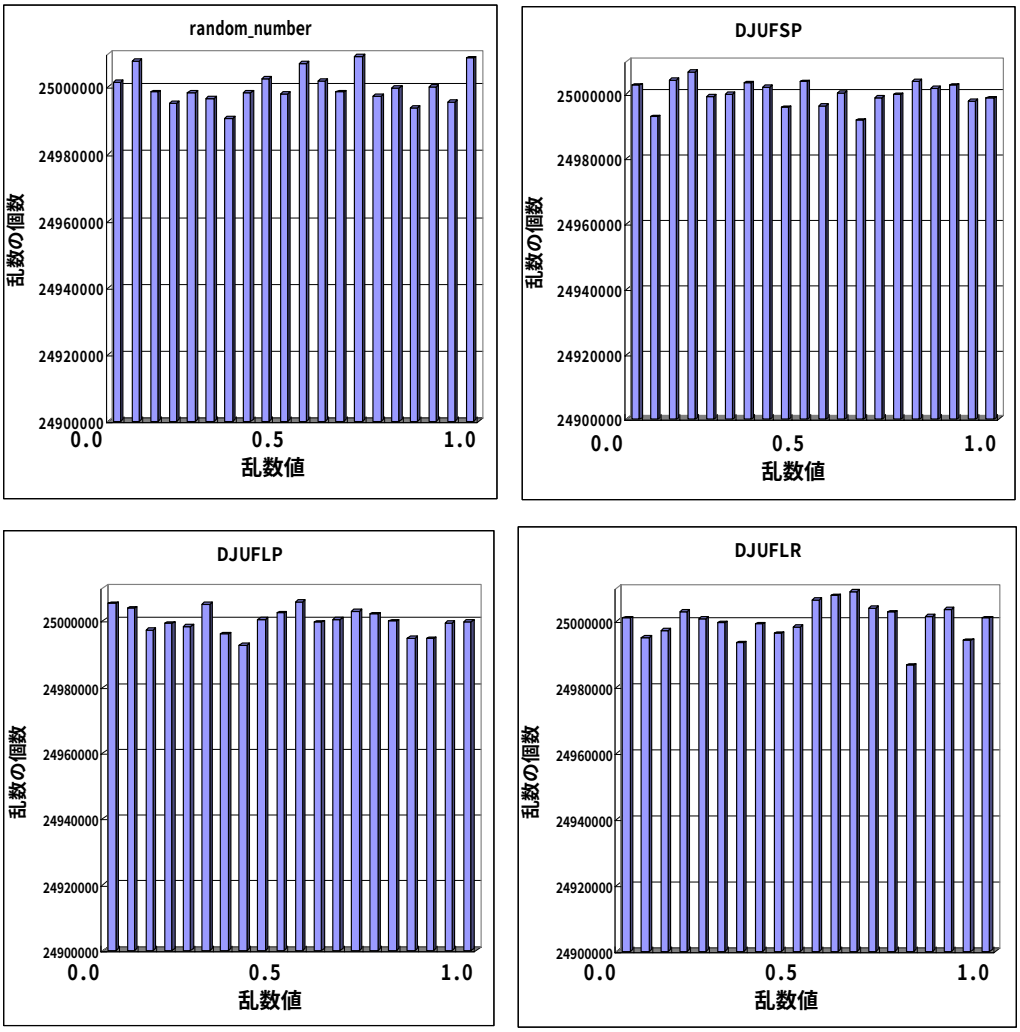

図 4.5.9.2　乱数の検定結果

ASL/SX の DJUFSP，DJUFLR については，並列処理機能版(それぞれ，QJUFSP,QJUFLR)が用意されているので，並列化されたコードで使用する時はそちらを使用されたい.

４．６　並列化チューニング事例

４．６．１　ループ入れ換えによる並列化

多重 do ループの場合，内側 do ループをベクトル化し，外側 do ループを並列化する．しかし，do ループ内に定義・参照のデータ依存関係があるため，ベクトル化は可能だが，並列化できないケースがある．このようなとき，do ループの入れ替えを行うことで，並列化できる場合がある．以下に，do ループの入れ替えを行うことで，並列化を促進する例を示す．

（１）性能の分析

図 4.6.1.1 に Proginf を示す. 図中①～④に示すように, ほとんどの部分が 1 つの CPU だけで実行されており，並列化できていないことがわかる．

```
      ******    プログラム   情報    ******
経過時間（秒）                :       321.469443
ユーザ時間（秒）              :       321.638126
システム時間（秒）            :         0.051050
ベクトル命令実行時間（秒）    :       321.381345
全命令実行数                  :     14232484592.
ベクトル命令実行数            :      8364659062.
ベクトル命令実行要素数        :   2137085789479.
浮動小数点データ実行要素数    :    854810382473.
MOPS 値                       :      6662.623121
MFLOPS 値                     :      2657.677413
MOPS 値    (実行時間換算)     :      6667.874798
MFLOPS 値 (実行時間換算)      :      2659.772272
平均ベクトル長                :       255.489886
ベクトル演算率（%）           :        99.726180
メモリ使用量（MB）            :       208.000000
最大同時実行可能プロセッサ数  :               4.
   1 台以上で実行した時間（秒）:       321.384801  ①
   2 台以上で実行した時間（秒）:         0.084926  ②
   3 台以上で実行した時間（秒）:         0.084758  ③
   4 台以上で実行した時間（秒）:         0.084139  ④
イベントビジー回数            :               0.
イベント待ち時間（秒）        :         0.000000
ロックビジー回数              :               0.
ロック待ち時間（秒）          :         0.000000
バリアビジー回数              :               0.
バリア待ち時間（秒）          :         0.000000
MIPS 値                       :        44.249992
MIPS 値（実行時間換算）       :        44.284871
命令キャッシュミス（秒）      :         0.002689
オペランドキャッシュミス（秒）:         0.009786
バンクコンフリクト時間（秒）  :         0.000026
```

図 4.6.1.1　Proginf

図 4.6.1.2 に ftrace 情報を示す．図中①に示すように，最もコストの高いサブルーチン subb は，並列化されていないことがわかる（並列化されている場合は，サブルーチン名の後ろに「$数字」が付く）．

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME    MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]                RATIO V.LEN    TIME   MISS    MISS      CONF

subb  ①       51200  321.369(100.0)     6.277 6668.0 2659.9 99.73 255.5  321.346  0.0010  0.0024  0.0000
suba               1    0.035(  0.0)    34.798 2088.1  179.9 96.82 116.0    0.035  0.0000  0.0000  0.0000
main               1    0.032(  0.0)    32.192  417.4   33.4  0.04 239.0    0.000  0.0004  0.0002  0.0000
suba$1             4    0.001(  0.0)     0.358 3685.3    0.1 98.96 255.8    0.001  0.0001  0.0001  0.0000
 -micro1           1    0.001(  0.0)     1.168 4511.2    0.0 99.11 255.8    0.001  0.0000  0.0000  0.0000
 -micro2           1    0.000(  0.0)     0.008   47.6    2.9  0.00   0.0    0.000  0.0000  0.0000  0.0000
 -micro3           1    0.000(  0.0)     0.107   30.6    0.2  0.00   0.0    0.000  0.0000  0.0000  0.0000
 -micro4           1    0.000(  0.0)     0.149   30.5    0.2  0.00   0.0    0.000  0.0000  0.0000  0.0000
subc               1    0.001(  0.0)     0.787  106.2    0.1 40.95 232.8    0.000  0.0003  0.0002  0.0000
-------------------------------------------------------------------------------------------------------------
total          51207  321.438(100.0)     6.277 6666.8 2659.3 99.73 255.5  321.381  0.0017  0.0029  0.0000
```

図 4.6.1.2　ftrace 情報

図 4.6.1.3 に編集リストを示す．図中の do ループは，外側の do ループの添え字 j に対して，定義・参照のデータ依存関係が生じており並列化されていない．

```
    3:                parameter(nx=1022,ny=1021)
      ...
    8: +------>       do j=1,ny
    9: |V----->         do i=1,nx
   10: ||                u(i,j)   = u(i,j)   - u1(i,j)*u2(i,j)/dd
   11: ||                v(i,j)   = v(i,j)   - v1(i,j)*v2(i,j)/dd
   12: ||                w(i,j)   = w(i,j)   - w1(i,j)*w2(i,j)/dd
   13: ||                e(i,j)   = e(i,j)   - e1(i,j)*e2(i,j)/dd
   14: ||                u(i,j+1) = u(i,j+1) + u1(i,j)*u2(i,j)/dd
   15: ||                v(i,j+1) = v(i,j+1) + v1(i,j)*v2(i,j)/dd
   16: ||                w(i,j+1) = w(i,j+1) + w1(i,j)*w2(i,j)/dd
   17: ||                e(i,j+1) = e(i,j+1) + e1(i,j)*e2(i,j)/dd
   18: |V-----         enddo
   19: +------       enddo
```

図 4.6.1.3　編集リスト

（2）並列化阻害要因の除去方法

外側 do ループの添え字 j については，並列化を阻害する依存関係は存在するが，ベクトル化を阻害するものではない．内側 do ループの添え字 i については，ベクトル化も並列化も阻害する依存関係はない．また，nx=1022，ny=1021 であり，２つの do ループは十分にベクトル長がある．

図 4.6.1.4 にソース修正後の編集リストを示す．内側と外側の do ループを入れ替えることにより，ベクトル化と並列化を行うことにした．またコンパイラは自動的に，配列の一次元目の添え字 i の do ループをベクトル化の対象とするように do ループの入れ替えを行うので，それを抑止するコンパイラ指示行を挿入した(図中の矢印).

```
 8: P------>         do i=1,nx
 9: |           !cdir noloopchg        ループ入れ換えの抑止
10: |V----->           do j=1,ny
11: ||                    u(i,j)   = u(i,j)   - u1(i,j)*u2(i,j)/dd
12: ||                    v(i,j)   = v(i,j)   - v1(i,j)*v2(i,j)/dd
13: ||                    w(i,j)   = w(i,j)   - w1(i,j)*w2(i,j)/dd
14: ||                    e(i,j)   = e(i,j)   - e1(i,j)*e2(i,j)/dd
15: ||                    u(i,j+1) = u(i,j+1) + u1(i,j)*u2(i,j)/dd
16: ||                    v(i,j+1) = v(i,j+1) + v1(i,j)*v2(i,j)/dd
17: ||                    w(i,j+1) = w(i,j+1) + w1(i,j)*w2(i,j)/dd
18: ||                    e(i,j+1) = e(i,j+1) + e1(i,j)*e2(i,j)/dd
19: |V-----             enddo
20: P------           enddo
```

図 4.6.1.4　ソース修正後の編集リスト

（3）ソース修正による効果

図 4.6.1.5 にソース修正後の Proginf を示す．図①～④に示すように，各 CPU が実行した時間に差がなく，並列化できていることがわかる．図中①に示すように実行時間も 121.0 秒に向上した（修正前は 321.4 秒）.

```
      ******    プログラム 情報    ******
経過時間（秒）                :       121.043601
ユーザ時間（秒）              :       483.995969
システム時間（秒）            :         0.051845
ベクトル命令実行時間（秒）    :       481.036157
全命令実行数                  :     14099358619.
ベクトル命令実行数            :      8372851062.
ベクトル命令実行要素数        :   2137085789479.
浮動小数点データ実行要素数    :    854810945670.
MOPS 値                       :      4427.335007
MFLOPS 値                     :      1766.153027
MOPS 値   (実行時間換算)      :     17703.545838
MFLOPS 値 (実行時間換算)      :      7062.300688
平均ベクトル長                :       255.239915
ベクトル演算率 (%)            :        99.732757
メモリ使用量 (MB)             :       208.000000
最大同時実行可能プロセッサ数  :               4.
   1 台以上で実行した時間（秒）:       121.038594  ①
   2 台以上で実行した時間（秒）:       121.007163  ②
   3 台以上で実行した時間（秒）:       121.006951  ③
   4 台以上で実行した時間（秒）:       120.943812  ④
  :
```

図 4.6.1.5　ソース修正後の Proginf

４．６．２　負荷バランスの調整

自動並列化が行われても，各 CPU に割り当てられた負荷バランスが不均等であると効率のよい並列化を行うことができない. 以下に, コンパイラ指示行の挿入により, 負荷バランスの不均等を解消した例を示す.

（1）性能の分析

図 4.6.2.1 に Proginf を示す. 図中①～④に示すように各 CPU が実行した時間にばらつきがあり，各 CPU に割り当てられた負荷バランスが不均等であることがわかる.

```
      ******    プログラム 情報    ******
経過時間（秒）                :       86.464324
ユーザ時間（秒）              :      228.643402
システム時間（秒）            :        0.350031
ベクトル命令実行時間（秒）    :      183.478110
全命令実行数                  :    12978973353.
ベクトル命令実行数            :     6482442272.
ベクトル命令実行要素数        :  1641243648398.
浮動小数点データ実行要素数    :   929323558981.
MOPS 値                       :     7206.594046
MFLOPS 値                     :     4064.510722
MOPS 値   (実行時間換算)      :    19077.496715
MFLOPS 値 (実行時間換算)      :    10759.686123
平均ベクトル長                :      253.182918
ベクトル演算率（%）           :       99.605731
メモリ使用量（MB）            :    18544.000000
最大同時実行可能プロセッサ数  :              4.
   1 台以上で実行した時間（秒）:       86.370880  ①
   2 台以上で実行した時間（秒）:       61.128696  ②
   3 台以上で実行した時間（秒）:       41.689851  ③
   4 台以上で実行した時間（秒）:       39.454017  ④
イベントビジー回数            :              0.
イベント待ち時間（秒）        :        0.000000
ロックビジー回数              :              0.
ロック待ち時間（秒）          :        0.000000
バリアビジー回数              :              0.
バリア待ち時間（秒）          :        0.000000
MIPS 値                       :       56.765134
MIPS 値（実行時間換算）       :      150.270246
命令キャッシュミス（秒）      :        0.032893
オペランドキャッシュミス（秒）:        1.532640
バンクコンフリクト時間（秒）  :        0.011514
```

図 4.6.2.1　Proginf

図 4.6.2.2 に ftrace 情報を示す．図中①～④に示すように，サブルーチン solve について，各 CPU の実行時間にばらつきがあることがわかる．

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]                RATIO V.LEN    TIME   MISS    MISS      CONF

solve$1         2048   220.987( 96.3)   107.904 7373.7 4198.1 99.63 256.0  175.160  0.0163  1.4971  0.0092
 -micro1  ①      512    41.531( 18.1)    81.115 8713.1 4964.0 99.70 256.0   38.971  0.0041  0.0057  0.0030
 -micro2  ②      512    55.307( 24.1)    08.022 6085.4 3461.8 99.55 256.0   36.138  0.0039  0.6680  0.0021
 -micro3  ③      512    59.403( 25.9)   116.022 6788.7 3863.6 99.60 256.0   43.320  0.0041  0.5630  0.0020
 -micro4  ④      512    64.745( 28.2)   126.456 8151.6 4642.5 99.67 256.0   56.732  0.0041  0.2605  0.0021
input              1     8.516(  3.7)  8515.812 2142.1  189.1 97.27 126.2    8.515  0.0009  0.0011  0.0000
solve            512     0.060(  0.0)     0.116   90.1    0.4 54.77 239.0    0.001  0.0107  0.0097  0.0017
main               1     0.001(  0.0)     0.840  157.6   12.9  0.00   0.0    0.000  0.0003  0.0001  0.0000
output             1     0.000(  0.0)     0.000    0.0    0.0  0.00   0.0    0.000  0.0000  0.0000  0.0000
-------------------------------------------------------------------------------------------------------------
total           2563   229.563(100.0)    89.568 7177.7 4048.2 99.61 253.2  183.677  0.0282  1.5080  0.0108
```

図 4.6.2.2　ftrace 情報

図 4.6.2.3 にサブルーチン solve の編集リストを示す．並列化対象の do ループ中に if 文があるため，if 文の真偽によって，各 CPU の演算の負荷にばらつきが出る．

```
    70: P------>         do k=1,nz
    71: |                  if(id(k).eq.0) then
    72: |W----->             do j=1,ny
    73: ||*---->               do i=1,nx
    74: |||                      v3(i,j,k) = R1*v1(i,j,k) + R2*v2(i,j,k) + R3 * R4
    75: |||                      u3(i,j,k) = R1*u1(i,j,k) + R2*u2(i,j,k) - R3 * R4
    76: |||                      w3(i,j,k) = R1*w1(i,j,k) + R2*w2(i,j,k) + R3 * R4
    77: ||*----                enddo
    78: |W-----              enddo
    79: |                  endif
    80: P------          enddo
```

図 4.6.2.3　編集リスト

（2）負荷バランスの調整方法

図 4.6.2.4 にサブルーチン solve を修正した編集リストを示す. 並列化対象となる do ループの分割方法を，コンパイラ指示行 concur(by=1)を使用して変更した.

```
69:            !cdir concur(by=1)
70: P------>         do k=1,nz
71: |                  if(id(k).eq.0) then
72: |W----->             do j=1,ny
73: ||*---->                do i=1,nx
74: |||                       v3(i,j,k) = R1*v1(i,j,k) + R2*v2(i,j,k) + R3 * R4
75: |||                       u3(i,j,k) = R1*u1(i,j,k) + R2*u2(i,j,k) - R3 * R4
76: |||                       w3(i,j,k) = R1*w1(i,j,k) + R2*w2(i,j,k) + R3 * R4
77: ||*----                 enddo
78: |W-----               enddo
79: |                    endif
80: P------           enddo
```

図 4.6.2.4　指示行挿入後の編集リスト

（3）分割方法変更による効果

図 4.6.2.5 に指示行挿入後の ftrace 情報を示す. 図中の①～④に示すようにサブルーチン solve の各 CPU の実行時間がほぼ均等になった．図 4.6.2.6 に指示行挿入後の Proginf を示す. 図中の①に示すように, 実行時間も 57.5 秒に短縮された(修正前は 86.4 秒).

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONF
solve$1	2048	195.437(95.8)	95.428	8334.6	4746.9	99.67	256.0	176.180	0.0107	0.7037	0.0287
-micro1 ①	512	48.878(24.0)	95.466	8778.2	5000.4	99.69	256.0	46.397	0.0026	0.0916	0.0075
-micro2 ②	512	48.860(24.0)	95.430	8121.8	4625.2	99.66	256.0	42.926	0.0024	0.2160	0.0071
-micro3 ③	512	48.847(23.9)	95.403	8210.6	4676.0	99.66	256.0	43.381	0.0028	0.1997	0.0071
-micro4 ④	512	48.852(24.0)	95.413	8227.8	4685.8	99.66	256.0	43.476	0.0028	0.1964	0.0071
input	1	8.493(4.2)	8492.617	2148.0	189.6	97.27	126.2	8.493	0.0000	0.0000	0.0000
solve	512	0.030(0.0)	0.059	149.0	0.7	64.84	239.0	0.001	0.0100	0.0092	0.0017
main	1	0.001(0.0)	0.830	159.5	13.0	0.00	0.0	0.000	0.0004	0.0001	0.0000
output	1	0.000(0.0)	0.001	226.0	21.0	0.00	0.0	0.000	0.0000	0.0000	0.0000
total	2563	203.961(100.0)	79.579	8075.8	4556.4	99.64	253.2	184.673	0.0211	0.7131	0.0305

図 4.6.2.5　指示行挿入後の ftrace 情報

```
      ******    プログラム  情報    ******
経過時間（秒）                :        57.549276
ユーザ時間（秒）              :       204.576108
システム時間（秒）            :         0.080975
   :
最大同時実行可能プロセッサ数  :               4.
   1 台以上で実行した時間（秒）:        57.544710  ①
   2 台以上で実行した時間（秒）:        49.042544
   3 台以上で実行した時間（秒）:        49.042457
   4 台以上で実行した時間（秒）:        48.946444
```

図 4.6.2.6　指示行挿入後の Proginf

４．６．３　並列化指示行による並列化（並列コンピュータ AzusA による高速化）

ベクトル演算率が低い場合は，SX-7 よりも AzusA の方が高速化が期待できる．ここでは AzusA による高速化の例を示す．

（1）性能の分析

図 4.6.3.1 に SX-7 で実行したときの ftrace 情報を示す．図中①に示すようにコストの 56.3%を占めているサブルーチン sub_a の②ベクトル演算率は 62.18%と低く，③平均ベクトル長も 40.1 と小さい．

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME    MOPS MFLOPS V.OP  AVER.   VECTOR  I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]              RATIO V.LEN    TIME    MISS    MISS      CONF
sub_a             2  1619.526( 56.3)809763.079  568.8   95.4 62.18  40.1  300.121  86.9083 256.0777  0.0033
                                 ①                          ②     ③
sub_b             2   520.635( 18.1)260317.737 7528.0 4277.8 99.52 246.1  520.440   0.0028   0.1089 90.7350
                       ④   ⑤                                 ⑥     ⑦                                ⑧
sub_c     781743062   316.736( 11.0)     0.000   66.6   12.3  0.00   0.0    0.000   0.0382  30.8979  0.0010
sub_d     335089316   283.083(  9.8)     0.001   66.3    0.0  0.00   0.0    0.000   0.0267  11.8700  0.0007
sub_e     111751232   128.940(  4.5)     0.001   72.8   10.4  0.00   0.0    0.000  29.0002   0.5667  0.0003
sub_f             2     5.561(  0.2)  2780.668 4300.5 1648.3 99.68 249.3    5.497   0.0000   0.0000  2.9719
main              1     0.796(  0.0)   795.889  184.3    3.6  0.27 242.9    0.000   0.1464   0.0186  0.0000
 …
-------------------------------------------------------------------------------------------------------------
total     1228606041  2875.304(100.0)     0.002 1709.0  833.4 91.52 148.7  826.059 116.1227 299.5404 93.7122
```

図 4.6.3.1　SX-7 の ftrace 情報

図 4.6.3.2 に sub_a の編集リストを示す．sub_a は 3 重 do ループで構成され，ベクトル化の対象となる do ループは最内側の 2 つである．2 つの do ループはループ長(n3,n4)が共に小さく，演算量も少ない．このため並列化が可能であればスカラ性能の高い AzusA での方が適していると思われる．

```
    1:                 subroutine sub_a
                         …
  569: +------>        do k=1,n1              ← ベクトル化されないループ
       |                 …
  601: |+----->          do j=1,n2            ← ベクトル化されないループ
  602: ||                 if(  ) then
  603: ||                     write(*,*) "error xxxxxx"
  604: ||                     stop
  605: ||                 endif
       ||                 …
  630: ||V---->           do i=1,n3            ← ベクトル化されるループ
    :  |||                  …
  635: ||V----            enddo
  636: ||V---->           do i=1,n4            ← ベクトル化されるループ
    :  |||                  …
  642: ||V----            enddo
       ||                 …
 1529: |+-----          enddo
       |                 …
 1547: +------         enddo
    :                    …
```

図 4.6.3.2　サブルーチン sub_a の編集リスト

一方, 図 4.6.3.1 中⑤に示すようにコストの 18.1%を占めるサブルーチン sub_b の⑥ベクトル演算率は 99.52%と高く，⑦平均ベクトル長も 246.1 と大きい．しかし，④実行時間 520.63 秒に対して⑧バンクコンフリクトが 90.73 秒も発生している．

図 4.6.3.3 に sub_b のソースを示す．sub_b では連立一次方程式（ガウスの消去法）を解いており，実行時間の大きいループで配列の二次元目を動かしている．これがバンクコンフリクトの原因であった. バンクコンフリクトを削減するには配列の一次元目を動かすようにプログラムを書き換える方法もあるが，ここではメーカ提供の数値計算ライブラリ ASL に置き換えることにした．また，sub_c，sub_d，sub_e は一回あたりの実行時間が短いが，呼び出し回数が非常に多いので，呼び出しのためのオーバヘッドを削減するためインライン展開を行うことにした.

```
      do i=1,n-1
            …
            do ii=i+1,n
                r=a(ii,i)/a(i,i)
                do j=i+1,n
                    a(ii,j)=a(ii,j)-r*a(i,j)
                enddo
                …
            enddo
              …
       enddo
```

図 4.6.3.3　サブルーチン sub_b のソース

（2）　並列化指示行の利用

図 4.6.3.2 で示したサブルーチン sub_a は，write 文(603 行目)があるため自動並列化の対象とならない．しかし，この write 文はパラメータが不適当な場合に実行され，プログラムが停止するように記述されているので，通常実行されることはない．そこで並列化指示行を用いて最外側の k のループ（569 行目）を並列化することとした.

図 4.6.3.4 に並列化指示行 paralleldo の追加とソース修正後のサブルーチン sub_a のリストを示す．ここで k のループの本体部分（図 4.6.3.2　570～1546 行）をサブルーチン化しているのはつぎの理由による．paralleldo を使用するときにはループ内でのみ使用する変数を並列化指示行で指定する必要がある．しかし，本体部分は約 1,000 行あり，多数の変数を使用しているので，それらを指定するのは大変である．そこでループの本体部分を別サブルーチン body とした（これにより，サブルーチン body で使用する変数はそれぞれの CPU ごとに確保される）.

```
subroutine  sub_a
 …
!cdir  paralleldo        ← 並列化指示行
do k=1,n1
  call body(k,…)         ← サブルーチン化した本体の呼び出し
enddo
   …
end

subroutine body(k,…)     ← ループ本体
   …
 do j=1,n2
       if(  ) then
          write(*,*) “error xxxxxxx”  } 不適当なパラメータ
          stop                        } の時，実行される
       endif
       …
       do i=1,n3
        …
       enddo
       do i=1,n4
       …
       enddo

   enddo
 return
 end
```

図 4.6.3.4　修正後のサブルーチン sub_a のソース

（3）　性能向上の結果

図 4.6.3.5 に性能向上の結果を示す. SX-7 では非並列で 2875.30 秒かかったものが, AzusA では，並列版 ASL，並列化指示行による並列化およびインライン展開を行った結果，16 並列時　319.59 秒となり，SX-7 に比べて約 9 倍の性能向上となった.

実行マシン	プログラム	非並列	16 並列	性能向上比
SX-7	オリジナル版	2875.30 秒	（1706.41 秒）	1
AzusA	改善版		319.59 秒	9

図 4.6.3.5　性能向上