［大規模科学計算システム］

SX-Aurora TSUBASA でのプログラミング（ベクトル化編）

岡野 進一　工藤 淑裕
日本電気株式会社

概 要

SX-Aurora TSUBASA システムのハードウエア性能を引き出すために重要となるプログラムのベクトル化、並列化に関わるコンパイラの機能、およびプログラミングの際にご留意頂きたい点について、「ベクトル化編」、「並列化編」の 2 回に分けてご紹介します。

SX-Aurora TSUBASA システムでは主なプログラミング言語として、Fortran、C、C++言語が利用できます。本稿では、SX-Aurora TSUBASA システムの Fortran 言語コンパイラである NEC Fortran コンパイラ（以降、単にコンパイラと略す）を用いて、コンパイラが持つ自動ベクトル化機能の特長、ベクトル化プログラミングでの性能向上についてご紹介します。

1. SX-Aurora TSUBASA の Vector Engine

今回、東北大学サイバーサイエンスセンターに導入された SX-Aurora TSUBASA B401-8 システムは、Linux OS 搭載の x86 ホストに、8 個のベクトルコアを内蔵した複数の Vector Engine(以下、VE)を搭載したビックコア、高メモリ帯域、省電力なスーパーコンピュータです。各ベクトルコアは一つの命令で最大 256 個のデータを処理できるベクトル演算器を備えています。

SX-Aurora TSUBASA のベクトル命令には、一般的なスカラプロセッサで導入されている複数のデータを一度に処理できる SIMD 命令に対して、次の特長があります。

- 最大 256 個までの大量のデータを一つのベクトル命令で演算できる
- 一つのベクトル命令で処理するデータの個数を自在に変更できる

SX-Aurora TSUBASA のベクトルコアのハードウエア性能を十分に引き出すためには、個々のベクトルコアの中でベクトル命令を使って効率的に計算するためのベクトル化が大変重要となります。

2. 自動ベクトル化機能

SX-Aurora TSUBASA システムでのベクトル化技法は、SX-ACE システムの場合と基本的には同じですが、今回はスーパーコンピュータを初めて使われる方にもご理解頂けるようベクトル化の基本概念からご紹介します。

2.1. ベクトル化の基本概念

通常の演算命令は、一度に一組のデータを演算できます。このような演算命令をベクトル命令と対比させるためにスカラ命令と呼びます。これに対してベクトル命令は、複数の組のデータに対する演算を一つの命

令で実行できます。

例-1 のプログラムは、配列 B と配列 C を加算し配列 A に代入、配列 E と配列 F を加算し配列 D に代入する DO ループです。

例-1

```
DO I = 1, 100
   A(I) = B(I) + C(I)
   D(I) = E(I) + F(I)
ENDDO
```

例-1 の DO ループ中の加算をスカラ命令、ベクトル命令で実行したときの実行イメージは図-1 のとおりです。一つのスカラ命令は、一度に一組のデータに対する演算処理を行います。これに対して、SX-Aurora TSUBASA のベクトル命令では一つの命令で一度に複数(この場合は 100 個、最大 256 個)のデータに対する演算処理を行うことができます。

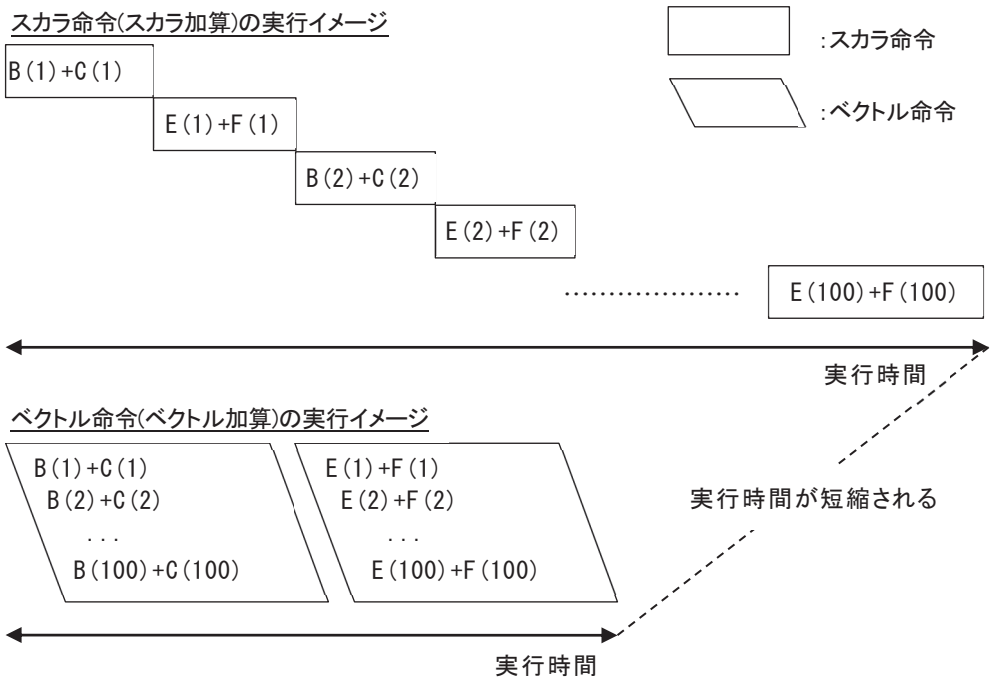


図-1 加算の実行イメージ

ループ中で計算される行列の要素など、規則的に並んだ配列データに対してベクトル命令を適用することをベクトル化(の適用)と呼び、ベクトル化することによって高速な演算が可能となります。

コンパイラの自動ベクトル化機能は、ソースプログラムを解析し、ベクトル命令で実行できる部分を自動的に検出しベクトル化を適用します。

2.2. ベクトル命令の適用例

例-2 は、DO ループが自動ベクトル化されたときのベクトル命令の適用例です。二つの配列のメモリロード、ベクトル加算、メモリストアの四つのベクトル命令で実行されます。

例-2

```
DO I = 1, 100
   C(I) = A(I) + B(I)
ENDDO

    ↓

VR1    ←  配列A          (配列Aからベクトルレジスタに100個のデータをロード)
VR2    ←  配列B          (配列Bからベクトルレジスタに100個のデータをロード)
VR3    ←  VR1 + VR2      (100個のデータをベクトル加算)
配列C ←  VR3            (配列Cに100個の演算結果をストア)
                    VRn : ベクトルレジスタ
```

2.3. ベクトル化の対象範囲

自動ベクトル化機能は、表-1 で示すループ、および、ループに含まれる文、データ、演算を対象としてベクトル化を適用します。

表-1　自動ベクトル化の対象

対象	Fortran 言語要素
ループ	配列式、DO ループ、DO WHILE ループ、FORALL ループ
文	代入文、CONTINUE 文、GOTO 文、CYCLE 文、EXIT 文、IF 文、SELECT 構文(CALL 文、入出力文等は不可)
データ型	INTEGER(KIND=4)、INTEGER(KIND=8)、REAL(KIND=4)、REAL(KIND=8)、COMPLEX(KIND=4)、COMPLEX(KIND=8)
演算	加減乗除算、べき算、論理演算、関係演算、型変換、組込み関数(利用者定義演算等は不可)

2.4. データの依存関係

ループにベクトル化を適用するには、「2.3. ベクトル化の対象範囲」で示した対象範囲に加えて、ベクトル化の適用による文、演算の実行順序が変更されても、データの定義・参照順番(データの依存関係)が変わらないことが条件となります。

例-3 は、二つの配列を定義する DO ループです。図-2 は、例-3 の DO ループのベクトル化適用前、すなわち、スカラでの文の実行順序とベクトル化後の文の実行順序のイメージを示しています。

例-3

```
DO I = 1, 100
   A(I) = B(I) + C(I)
   D(I) = E(I) + F(I)
ENDDO
```

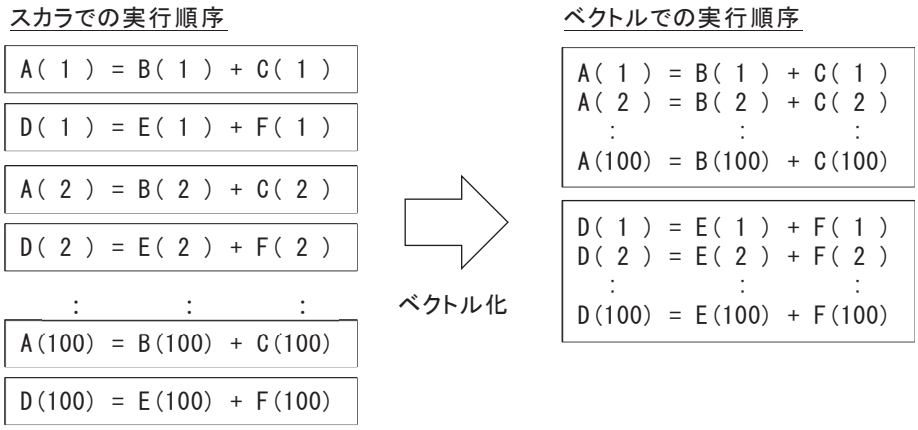


図-2　文の実行順序

スカラでの実行では、A(1)=B(1)+C(1)、D(1)=E(1)+F(1)、A(2)=B(2)+C(2)、...と実行されますが、ベクトル化した場合は一つの命令で複数のデータを処理するため、A(1)=B(1)+C(1)、A(2)=B(2)+C(2)、...、A(100)=B(100)+C(100)、D(1)=E(1)+F(1)、D(2)=E(2)+F(2)、...、D(100)=E(100)+F(100)と配列 B、配列 C の加算、配列 A への代入後、配列 E、配列 F の加算、配列 D への代入が実行されます。ベクトル化した場合、このように文、演算の実行順序が変わります。

このような文の実行順序の変更により、データの依存関係が変わってしまうことがあります。例-4 はベクトル化後にデータの依存関係が変わってしまう例です。以前の繰返しで定義された配列要素や変数を後の繰返しで参照するパターンのとき、図-3 のようにデータの依存関係が変わり正しい結果が得られなくなるため、コンパイラはベクトル化を適用しません。

例-4

```
DO I = 2, N
   A(I+1) = A(I) * B(I) + C(I)
ENDDO
```

図-3 は、例-4 の DO ループの文の実行順序示すイメージです。ベクトル化すると、「2.2. ベクトル命令の適用例」で示した例-2 のように、配列 A のデータをまとめてメモリからロードするため、更新された A の値が使用されず正しい演算結果が得られません。

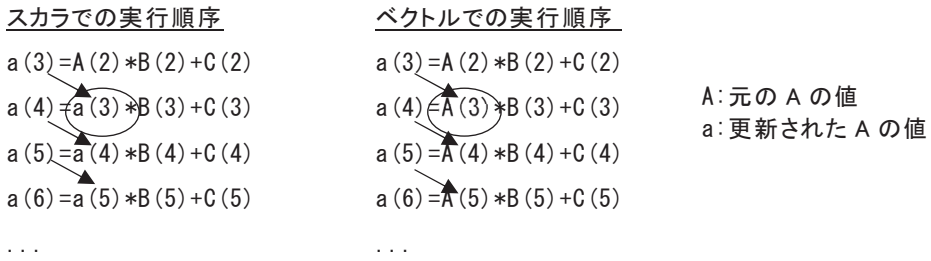


図-3　ベクトル化を阻害するデータの依存関係

例-5 はベクトル化を適用してもデータの依存関係が変わらない例です。

例-5

```
DO I = 2, N
   A(I-1) = A(I) * B(I) + C(I)
ENDDO
```

図-4 は、例-5 の DO ループの文の実行順序を示すイメージです。ベクトル化してもデータの依存関係は変わらないのでコンパイラはベクトル化を適用します。

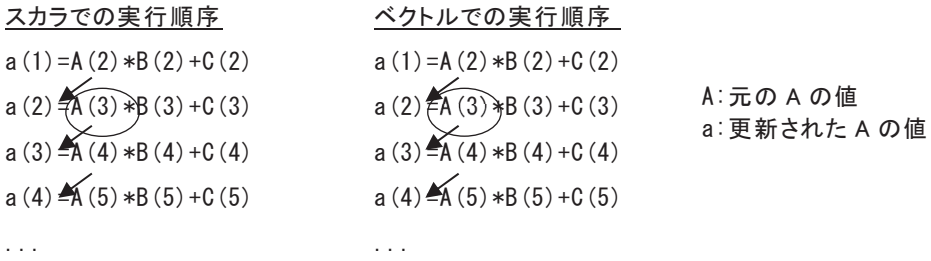


図-4　ベクトル化可能なデータの依存関係

コンパイラの自動ベクトル化機能は、ソースプログラムを解析して、ベクトル命令で実行できる部分を自動的に検出するとともに、必要ならベクトル化に適合するようにプログラムを変形して、ベクトル化を適用できる範囲を広げます。

プログラムのループを記述する際、文をループの繰返しごとに並べたとき、図-3 のような右下向きの矢印の依存関係ができないようにすると、ループにベクトル化を適用でき、ループ内の処理を高速化できる可能性が高まります。

3.　拡張ベクトル化機能

コンパイラの自動ベクトル化機能は、ベクトル化を適用できる範囲を拡げるため、データの依存関係などの理由でベクトル化できないループを変形してベクトル化したり、プログラムを変形することによってベクトル

化の効果をさらに高めたりします。これを拡張ベクトル化機能と呼びます。ここでは、コンパイラの持つ拡張ベクトル化機能のうち主なものをご紹介します。

3.1. 文の入れ換え

「2.4 データの依存関係」の例のようなループは、ループ中の文を入れ換えるとベクトル化できることがあります。コンパイラは、実行結果が正しく保てるのであれば、例-6 に示すように自動的に文を入れ換えてベクトル化を適用します。

例-6

（文の入れ換え前）

```
DO I = 1, 99
   A(I) = 2.0      ! 定義
   B(I) = A(I+1)   ! 参照
ENDDO
```

（文の入れ換え後のイメージ）

```
DO I = 1, 99
   B(I) = A(I+1)       ! 参照
   A(I) = 2.0          ! 定義
ENDDO
```

3.2. ループの一重化

ループの一重化は、多重ループの外側のループと内側のループを一つのループにまとめて、ループの繰返し数を大きくする最適化です。

SX-Aurora TSUBASA のベクトル命令は一つの命令で最大 256 個のデータを処理できます。よって、ベクトル命令で処理するときはできるだけ 256 個ずつ演算した方が命令の実行回数を減らすことができ効率がよくなります。さらに外側ループの繰返し制御のための時間も省くことができます。

自動ベクトル化では最内側ループをベクトル化の対象とします。例-7 では最内側ループの繰返し数は最大でも 100 回と 256 より短いので、このままでは一度に 100 個ずつ処理するようベクトル化されてしまいます。このループに一重化を適用すると、繰返し数を 10,000(=100×100)にでき、256 個ずつ処理できるようになります。

例-7

（ループの一重化前）

```
INTEGER::M, N
PARAMETER(M=100, N=100)
REAL(KIND=8)::A(M,N), B(M,N), C(M,N)
DO I = 1, M
   DO J = 1, N
      A(J,I) = B(J,I) + C(J,I)
   ENDDO
ENDDO
```

（ループの一重化後の変形イメージ）

```
DO IJ = 1, M*N
   A(IJ,1) = B(IJ,1) + C(IJ,1)
ENDDO
```

3.3. ループの入れ換え

多重ループのとき、ループを入れ換えることによりベクトル化できないデータの依存関係の問題が解消されてベクトル化できるようになる場合や、内側のループよりも外側のループの繰返し数が大きく、入れ換えた方が効率がよいと判断された場合には、例-8 のようにコンパイラが自動的にループを入れ換えてベクトル化を適用します。

例-8

（ループの入れ換え前）

```
DO J=1, 1000
   DO I = 1, 999
      A(I+1,J) = A(I,J) + B(I,J)
   ENDDO
ENDDO
```

（ループの入れ換え後のイメージ）

```
DO I = 1, 999
   DO J = 1, 1000
      A(I+1,J) = A(I,J) + B(I,J)
   ENDDO
ENDDO
```

なお、入れ換えた際の効率は、ハードウエア特性、およびループ内での処理内容や配列要素のアクセス方法(連続アクセス、飛びアクセス)など、さまざまな要因により変わります。このため、SX-ACE ではループ入れ換えされていたループが、SX-Aurora TSUBASA ではループ入れ換えの対象とならない場合があります。

3.4. 条件ベクトル化

条件ベクトル化とは、一つのループに対してあらかじめ二つ以上の命令コードを用意しておき、実行時に最も効率よく実行できる命令コードを選択して実行するベクトル化です。

例-9 は、データの依存関係がベクトル化に適合しているかどうかコンパイル時に不明であったとき、ベクトル化した命令コードとベクトル化しない命令コードの両方を用意しておき、プログラムを実行するときそのどちらかを選択して実行する条件ベクトル化の例です。

例-9

（条件ベクトル化前）

```
DO I = N, N+10
   A(I) = A(I+K) + B(I)
ENDDO
```

（条件ベクトル化後のイメージ）

```
IF (K.GE.0.OR.K.LT.-10) THEN
!NEC$ IVDEP              ! ベクトルでの実行
   DO I = N, N+10
      A(I) = A(I+K) + B(I)
   ENDDO
ELSE
   DO I = N, N+10        ! スカラでの実行
      A(I) = A(I+K) + B(I)
   ENDDO
ENDIF
```

例-9 の条件ベクトル化後のイメージの IF 文が実行時にコードを選択するための判定文で、THEN ブロックはベクトルで実行するコード、ELSE ブロックはスカラで実行するコードです。IF 文の条件式に現れる

「K.GE.0」が真のときは例-5 と同じようにデータの依存関係が変わりません。また、「K.LT.-10」のときには、「DO I=N,N+10」であることから常に I≠I+K が成り立ちデータの依存関係がありません。よって、条件式が真のときベクトルでの実行(THEN ブロック)となります。逆に、IF 文の条件式が偽であるとき、「2.4 データの依存関係」の例-4、図-3 で示したようなベクトル化後にデータの依存関係が変わってしまう(右下向きの矢印ができてしまう)のでスカラでの実行(ELSE ブロック)となります。

例-9 はデータの依存関係に着目した条件ベクトル化(依存関係による条件ベクトル化)です。他の条件ベクトル化として、ループの繰返し数による条件ベクトル化も行います。

3.5. マクロ演算の認識

次のようなパターンは、変数や配列要素が繰り返しにまたがって定義・引用されるため、本来はベクトル化できませんが、コンパイラが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を用いることで，ベクトル化を行います。

例-10 総和

```
SUM = 0.0
DO I=1, N
   SUM = SUM + A(I)
END DO
```

例-11 最大値・最小値

```
DO I=1, N
   IF (XMAX .LT. X(I)) THEN
     XMAX = X(I)
   END IF
END DO
```

例-12 圧縮・伸長

```
J = 0
DO I=1, N
   IF (X(I) .GT. 0.0) THEN
      J = J + 1
      Y(J) = Z(I)
   END IF
 END DO
```

例-13 漸化式

```
DO I=1, N
    A(I) = A(I-1) * B(I) + C(I)
END DO
```

4. 基本的な使い方

この章では、コンパイラの一般的な使い方について説明します。

4.1. コンパイラの起動

SX-Aurora TSUBASA システム用の Fortran コンパイラのコマンドは nfort です。一般的なコンパイラ同様、例-14 のように使用します。

例-14

$ nfort [コマンドラインオプション…] 入力ファイル…

4.2. コマンドラインオプション

代表的なコマンドラインオプションは、GNU コンパイラなど一般的なコンパイラと同じなので、Makefile などを大きく修正することなく使用できます。一方、SX-ACE システム用のコンパイラのコマンドラインオプションからは多くが変更されているため、ユーザーズガイドの付録 B の対応表を参考にして、適宜修正してください。

以下では、最適化やチューニングにおける主なコマンドラインオプションについて紹介します。

-O*n*

自動ベクトル化、最適化のレベルを指定します。各レベルの説明を表-2 に示します。

表-2　自動ベクトル化のレベル

-O4	最大レベルの自動ベクトル化を適用
-O3	高度なレベルの自動ベクトル化を適用
-O2	既定レベルの自動ベクトル化を適用（既定値）
-O1	副作用のない自動ベクトル化を適用
-O0	ベクトル化、最適化を適用しない

-mparallel

自動並列化機能を適用します。詳細は、「並列化編」でご紹介します。

-fopenmp

OpenMP 機能を適用します。詳細は、「並列化編」でご紹介します。

-finline-functions

自動インライン展開機能を適用します。

-report-all

オプションリスト、診断メッセージリスト、編集リストを出力します。

オプションリストには、コンパイラオプションのコンパイル時の状態(有効、無効、値など)が出力されます。

例-15　　オプションリストの例

```
NEC Fortran Compiler (3.0.7) for Vector Engine Mon Aug 17 13:25:29 2020
FILE NAME: fft.f90

  COMPILER OPTIONS : -report-option

OPTIONS DIRECTIVE: -O4

PARAMETER :

Optimization Options :
  ....
  -On                          : 4
  -fargument-alias             : disable
  -fargument-noalias           : enable
  -fassociative-math           : enable
  ....
```

診断メッセージリストや編集リストはプログラムに対してコンパイラがどのような最適化を適用したか、および、適用できなかった理由等が出力されます。診断メッセージリストは、コンパイル時に標準エラー出力に出力された診断メッセージがリストとなって出力されます。編集リストは以下のような形式で出力されます。

例-16　　編集リストの例

```
NEC Fortran Compiler (3.0.7) for Vector Engine Mon Aug 17 15:00:01 2020
FILE NAME: a.f90

PROCEDURE NAME: SUB
FORMAT LIST

 LINE   LOOP   STATEMENT
```

```
1:            SUBROUTINE SUB(A, B, N, M)
2:            INTEGER::N, M
3:            REAL(KIND=8)::A(M, N), B(M, N)
4:    +------>  DO J=1,M
5:    |V----->    DO I=1, N
6:    ||            A(I,J) = A(I,J) + B(I,J)
7:    |V-----      ENDDO
8:    +------   ENDDO
9:            END SUBROUTINE
```

4.3. コンパイラ指示行

コンパイラは指定されたコンパイルオプションに応じて、最適なベクトル化、および最適化を適用しますが、コンパイル時にソースプログラムからコンパイラが自動で認識できない情報(変数の値やループの繰り返し数など)のため、本来適用すべきベクトル化や最適化ができない場合があります。そのような場合、コンパイラ指示行を指定することにより、コンパイル時に必要な情報を補い、ベクトル化や最適化を適用できるようになります。

IVDEP 指示行

コンパイル時にコンパイラが自動的に依存関係を解析しループの自動ベクトル化を試みますが、コンパイル時に依存関係が不明な場合、ループを自動ベクトル化しません。このとき、プログラマがプログラムに依存関係がないことが分かっている場合に IVDEP 指示行を指定すると、コンパイラはベクトル化が不可となる依存関係はないと仮定してベクトル化を試みます。

例-17

```
SUBROUTINE SUB(A,B,C,N,K)
   REAL::A(N), B(N), C(N)
   INTEGER::N,K,I

!NEC$ IVDEP
   DO I=1,N
      A(I+K) = A(I) + B(I)
   END DO

END SUBROUTINE SUB
```

NOVECTOR 指示行

自動ベクトル化の適用から除外するためには、NOVECTOR 指示行を使用します。依存関係はなく、自動ベクトル化は行われたが、実際のループの繰り返し数が非常に少なく、ベクトル化しない場合の方が高速な場合やベクトル化による演算結果の誤差が気になる場合などに使用します。

例-18

```
SUBROUTINE SUB(A,B,C,N,K)
   REAL:: A(N), B(N), C(N)
   INTEGER::N,K,I

!NEC$ NOVECTOR
   DO I = 1, M
      A(I+K) = A(I) * B(I)
   ENDDO

END SUBROUTINE SUB
```

OUTERLOOP_UNROLL 指示行

多重ループの場合、外側のループに対して、OUTERLOOP_UNROLL 指示行を指定することで、外側のループを内側のループ内部に展開(アンロール)することにより、内側ループのインデックスのみを使用するロードやストアの命令の実行数を減らすことができ、高速化できます。

例-19

```
!NEC$ OUTERLOOP_UNROLL(4)
   DO J = 1, M
      DO I = 1, N
         A(I,J) = B(I,J) + C(I)     ! C(I)のロードが1/4に減る
      ENDDO
   ENDDO
```

nfdirconv コマンド

コマンドラインオプションと同様、多くのコンパイラ指示行も、SX-ACE 用のコンパイラである FORTRAN90/SX や Fortran 2003 コンパイラのコンパイラ指示行から変更されています。

nfdirconv コマンドは、SX-ACE システム用のコンパイラである FORTRAN90/SX や Fortran 2003 コンパイラ用に記述されたコンパイラ指示行を、SX-Aurora TSUBASA システム用の Fortran コンパイラのコンパイラ指示行に変換するツールです。

使い方は、例-20 のとおりで、入力ファイルの代わりにディレクトリを指定した場合、ディレクトリ配下のすべてのファイルのコンパイラ指示行をまとめて変換できます。

例-20

```
$ nfdirconv [オプション …] 入力ファイル …
```

4.4. ベクトル化による演算結果への影響

この節では、SX-Aurora TSUBASA においてプログラミングを行う際に留意すべき点として、ベクトル化の最適化を行った場合とベクトル化を行わなかった場合で、演算結果が誤差範囲で異なる場合があります。

- 最適化・ベクトル化による演算順序の変更や除算の乗算化により、情報落ちや桁落ち、丸め誤差などが変わるため
- ベクトル化された数学関数では、高速にベクトル計算できるようスカラ版の数学関数と異なる計算アルゴリズムを使用しているため
- ベクトル積和演算(fused multiply-add, FMA)が使用された場合、途中の積算結果を丸めずに加算が行われるため、使用しない場合に対して異なる演算結果となる可能性があるため

このような誤差が気になる場合には、NOVECTOR 指示行を指定してループがベクトル化されないようにしたり、NOFMA 指示行を指定してベクトル融合積和演算が行われないようにしてください。

4.5. 実行時環境変数

この節では、プログラムの実行時に便利ないくつかの実行時環境変数を説明します。

VE_FORT_UFMTENDIAN

SX-Aurora TSUBASA は SX-ACE と異なりリトルエンディアンのシステムです。SX-ACE などのビッグエンディアンのシステムの外部ファイルを書式なし入出力するとき、エンディアン変換が必要となります。そのようなとき、この環境変数で、ビッグエンディアンの外部ファイルが接続されている装置番号を指定します。

例-21　すべての外部ファイル装置に対してビッグエンディアンとする指定

```
$ export VE_FORT_UFMTENDIAN=ALL
```

例-22　外部ファイル装置 10、11に対してビッグエンディアンとする指定

```
$ export VE_FORT_UFMTENDIAN=10,11
```

例-23　外部ファイル装置 10、11、12 に対してビッグエンディアンとする指定

```
$ export VE_FORT_UFMTENDIAN=10-12
```

例-24　外部ファイル装置 10、11、12 以外をビッグエンディアンとする指定

```
$ export VE_FORT_UFMTENDIAN=big;little:10-12
```

VE_FORT_EXPRCW

2GB(ギガバイト)を超える記録(レコード)を扱うために拡張されたフォーマットを使用する書式なし入

出力の外部ファイル装置を指定します。

例-25　　外部ファイル装置 10 に対して指定

```
$ export VE_FORT_EXPRCV=10
```

5. 性能解析

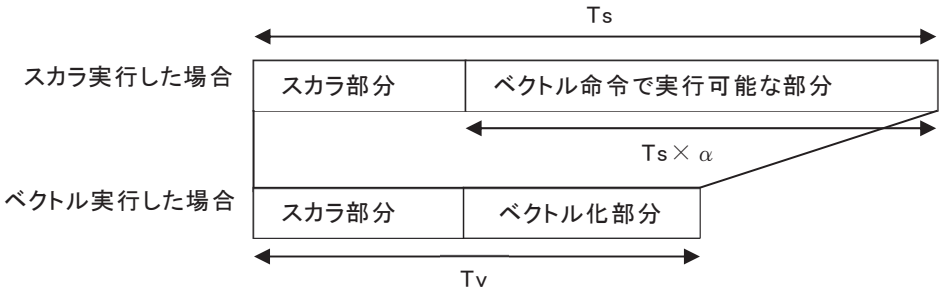
SX-Aurora TSUBASA においては、プログラムに含まれるループをできるだけベクトル命令を使って、データを最大ベクトル長である 256 個ずつ実行できているかどうかが高速化のポイントです。

本セクションでは、プログラムの性能を分析するための性能値、その取得方法、ベクトル化が十分できていない手続を絞り込む方法などを説明します。

5.1. ベクトル化率

プログラムをスカラ命令だけで実行させた場合の実行時間に占めるベクトル命令で実行可能な部分の時間の割合をベクトル化率と呼びます。一般にベクトル化率を正確に求めることは困難であるため、ベクトル化率の近似値としてベクトル演算率を用います。

ベクトル演算率は、プログラムで処理された全演算数に占めるベクトル命令で処理された演算数の割合を求めたものです。SX-Aurora TSUBASA では、このベクトル演算率を大きくすることを目標にチューニングしてください。ベクトル演算率は、プログラム中のループをベクトル化することにより実行性能が向上しますが、ベクトル演算率が 50%程度では、スカラ実行時の高々2 倍の性能にしかならないことは、図-5 からもわかると思います。一般にはベクトル演算率が 98%以上を目指すことになります。ベクトル演算率は、以降で示すプログラム実行解析情報、簡易性能解析機能/Ftrace Viewer で参照できます。



Ts：スカラ実行したときの実行時間　　α：ベクトル化率

Tv：ベクトル実行したときの実行時間

図-5 ベクトル化率

5.2. プログラム実行解析情報

プログラム実行解析情報は、プログラムの実行時に参照される環境変数 VE_PROGINF に YES、または、DETAIL が設定されているとき、プログラムの実行終了時に標準エラー出力ファイルに出力されます。この情報から、プログラムがハードウエアの性能を十分に引き出しているか否かを判断できます。

例-26 にプログラム実行解析情報の出力例を示します。

例-26

```
            ******  Program Information  ******
Real Time (sec)                 :          204.076110   経過時間
User Time (sec)                 :          203.706817   ユーザ時間
Vector Time (sec)               :          197.623752   ベクトル命令実行時間
Inst. Count                     :        38596814372    全命令実行数
V. Inst. Count                  :        13465836887    ベクトル命令実行数
V. Element Count                :      2957231889428    ベクトル命令実行要素数
V. Load Element Count           :       997524789907    ベクトル命令ロード要素数
FLOP Count                      :      1776569208614    浮動小数点データ実行要素数
MOPS                            :          18087.515129 MOPS値(ユーザ時間)
MOPS (Real)                     :          18053.533350 MOPS値(経過時間)
MFLOPS                          :           8721.924006 MFLOPS値(ユーザ時間)
MFLOPS (Real)                   :           8705.537759 MFLOPS値(経過時間)
A. V. Length                    :            219.609959 平均ベクトル長
V. Op. Ratio (%)                :             99.317880 ベクトル演算率
L1 Cache Miss (sec)             :              5.637238 L1キャッシュミス時間
CPU Port Conf. (sec)            :              0.125939 CPUポート競合時間
V. Arith Exec. (sec)            :             29.765092 ベクトル演算実行時間
V. Load Exec. (sec)             :            163.530245 ベクトルロード実行時間
VLD LLC Hit Element Ratio (%)   :             58.115252 ベクトル要素LLCヒット率
Power Throttling (sec)          :              0.000000 電力要因HW停止時間
Thermal Throttling (sec)        :              0.000000 温度要因HW停止時間
Memory Size Used (MB)           :           5376.000000 最大メモリ使用量
```

プログラム実行解析情報のベクトル演算率が 98%未満のとき、プログラムを調べ、ベクトル化できていないループがないかなどを調査します。

5.3. 簡易性能解析機能／Ftrace Viewer

プログラムの規模が大きいとき、プログラム内のコードすべてについてループのベクトル化状況を調べるのは作業効率がよくありません。ベクトル性能を引き出すには、実行時間が長く、ベクトル演算率が低い手続に絞って調べることが効果的です。この絞り込みには簡易性能解析機能、または Ftrace Viewer が有効です。

Ftrace Viewer では、手続ごとの実行回数、実行時間、MFLOPS 値、ベクトル演算率、LLC ヒット要素率などの様々な性能値に加えて、それらをグラフ(Function Metric Chart)表示し、視覚的にプログラムの性能を分析できます。

図-6 は、実行時間とベクトル演算率を重ねて表示したグラフです。これを参照し、実行時間が長く、ベクトル演算率の低いものを探し、その中のループのベクトル化状況を調べます。

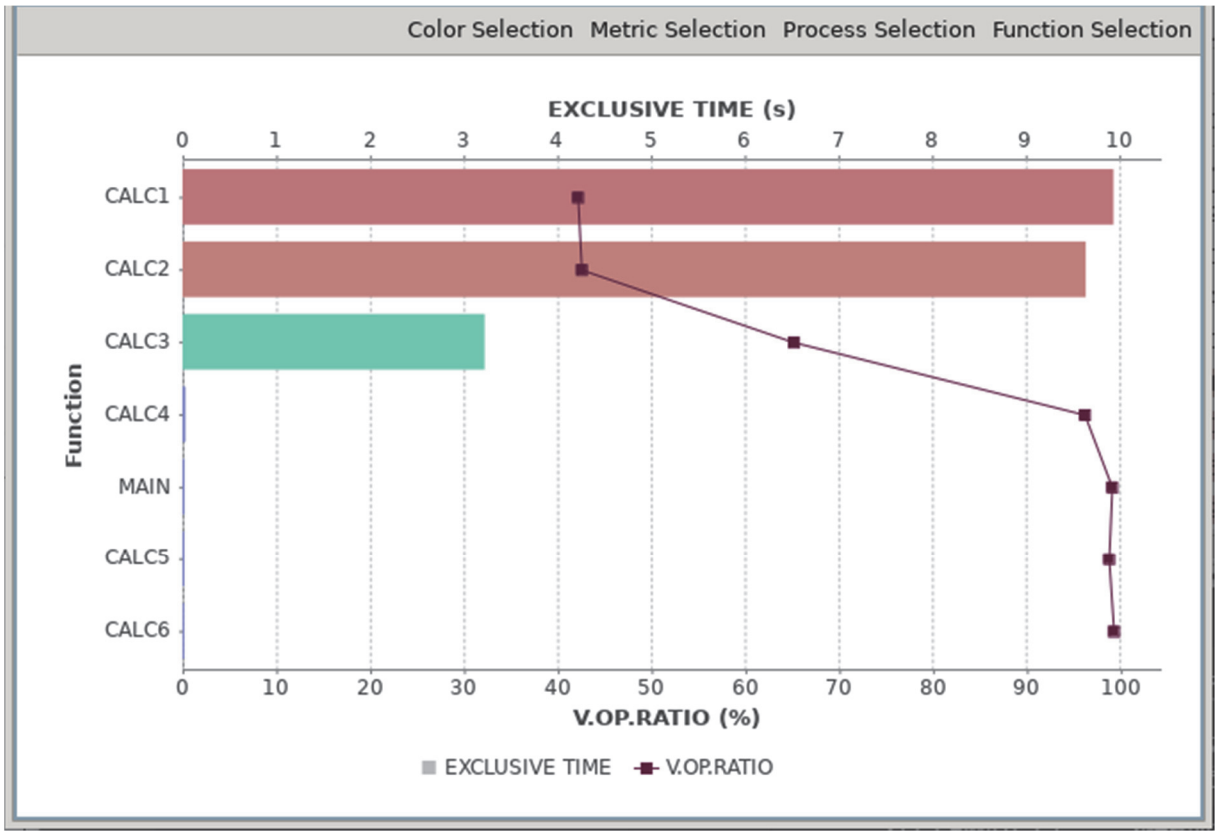
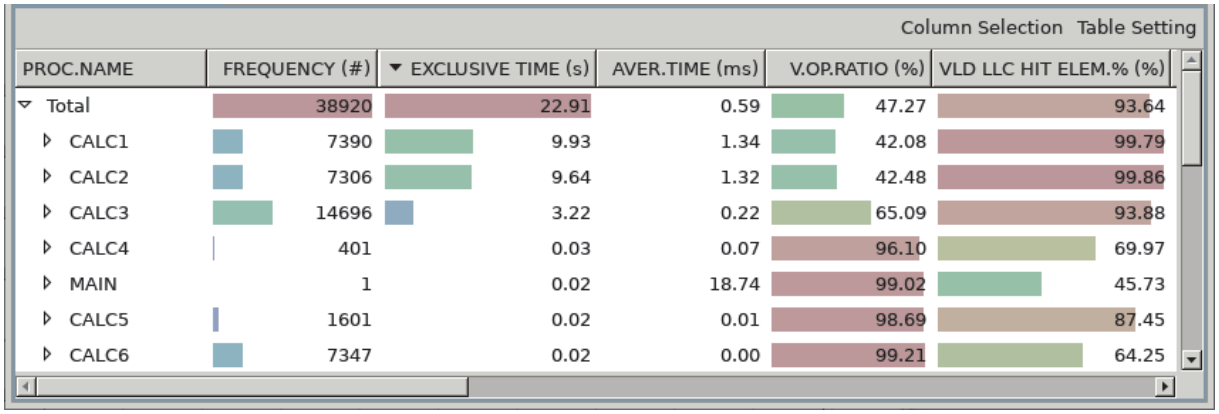


図-6 Function Metric Chart

図-6 では、縦軸が手続名、上横軸が実行時間(EXCLUSIVE TIME、単位は秒)、下横軸がベクトル演算率(V.OP.RATIO、単位は%)です。実行時間は棒グラフ、ベクトル演算率は折れ線グラフで表示されています。

このグラフから、実行時間が長くベクトル演算率が低い手続は「calc1」と「calc2」であることが分かります。「calc1」も「calc2」も実行時間が長く、かつ、ベクトル演算率も低いため、ベクトル演算率を高めるという観点からのチューニングにより高速化が期待できます。また、「calc3」についてもベクトル演算率が十分に高いとは言えないため、チューニングの余地はありそうです。その他の手続については、ベクトル演算率も高く、実行時間も非常に短くなっているため、これ以上のチューニングの必要はなさそうです。

個々のサブルーチン、手続の詳細な性能値を調べたいとき、グラフと一緒に表示されるテーブル(Function Table)を参照します。



PROC.NAME	FREQUENCY (#)	EXCLUSIVE TIME (s)	AVER.TIME (ms)	V.OP.RATIO (%)	VLD LLC HIT ELEM.% (%)
Total	38920	22.91	0.59	47.27	93.64
CALC1	7390	9.93	1.34	42.08	99.79
CALC2	7306	9.64	1.32	42.48	99.86
CALC3	14696	3.22	0.22	65.09	93.88
CALC4	401	0.03	0.07	96.10	69.97
MAIN	1	0.02	18.74	99.02	45.73
CALC5	1601	0.02	0.01	98.69	87.45
CALC6	7347	0.02	0.00	99.21	64.25

図-7　Function Table

図-7 から、手続「calc1」の実行回数(FREQUENCY)は 7390 回、一回当たりの実行時間(AVER.TIME)は 1.34 ミリ秒で総実行時間は 9.93 秒だったことがわかります。また、手続「calc2」は実行回数(FREQUENCY)は 7306 回、一回当たりの実行時間(AVER.TIME)は 1.34 ミリ秒で総実行時間は 9.64 秒だったことがわかります。さらに、ベクトル演算率(V.OP.RATIO)は手続「calc1」が 42.08%、手続「calc2」が 42.48%だったこともわかります。

Ftrace Viewer で性能値を参照するには、コンパイラオプション-ftrace を指定してコンパイルされたプログラムを実行して、解析情報ファイル(ファイル名 ftrace.out.*.*)を出力することが必要です。

6.　おわりに

以上、NEC Fortran コンパイラの自動ベクトル化機能を中心にご紹介させて頂きました。SX-Aurora TSUBASA のコンパイラは、他にも本稿でご紹介できなかった種々のベクトル化機能を持っています。詳細につきましては、マニュアル「SX-Aurora TSUBASA FORTRAN コンパイラ ユーザーズガイド」をご参照ください。

皆様が SX-Aurora TSUBASA をご利用になる上で、本稿が多少なりともお役に立てれば幸いです。

参考文献

[1] NEC Aurora Forum　https://www.hpc.nec/

[2] SX-Aurora TSUBASA Fortran コンパイラ ユーザーズガイド 日本電気株式会社 G2AF02
https://www.hpc.nec/documentation

[3] SX-Aurora TSUBASA Ftrace Viewer ユーザーズガイド 日本電気株式会社 G2AT01
https://www.hpc.nec/documentation