ISSN 2436-0066

TOHOKU
UNIVERSITY

東北大学
サイバーサイエンスセンター

大規模科学計算システム広報

SENAC

Vol.56 No.4 2023–10

Cyberscience
Center

Supercomputing System
Tohoku University
www.ss.cc.tohoku.ac.jp

大規模科学計算システム関連案内

＜大規模科学計算システム関連業務は、サイバーサイエンスセンター本館内の情報部情報基盤課が担当しています。＞

https://www.ss.cc.tohoku.ac.jp/

階	係・室名	電話番号(内線)* e-mail	主なサービス内容	サービス時間 平 日
一階	利用相談室	022-795-6153(6153) 相談員不在時 022-795-3406(3406)	計算機利用全般に関する相談 大判プリンタ、利用者端末等の利用	8:30～17:15 9:00～21:00
	利用者談話室	(3444)	自販機	8:30～21:00
	展 示 室*（分散コンピュータ博物館)*	*見学希望の方は共同利用支援係までご連絡ください。	歴代の大型計算機等の展示	9:00～16:00
三階	総務係	022-795-3407(3407) cc-som@grp.tohoku.ac.jp	総務に関すること	8:30～17:15
	会計係	022-795-3405(3405) cc-kaikei＠grp.tohoku.ac.jp	会計に関すること、負担金の請求に関すること	8:30～17:15
	共同利用支援係 （受 付）	022-795-3406(3406) 022-795-6251(6251) cc-uketuke@grp.tohoku.ac.jp	利用手続き、利用相談、講習会、ライブラリ、見学、アプリケーションに関すること	8:30～17:15
	共同研究支援係	022-795-6252(6252) rs-sec@cc.tohoku.ac.jp	共同研究、計算機システムに関すること	8:30～17:15
	ネットワーク係	022-795-6253(6253) i-network@grp.tohoku.ac.jp	ネットワークに関すること	8:30～17:15
四階	研究開発部	022-795-6095(6095)		
五階	端末機室	(3445)	PC 端末機(X 端末)	

＊（ ）内は東北大学内のみの内線電話番号です。青葉山・川内地区以外からは頭に 92 を加えます。

本誌の名前「SENAC」の由来

昭和 33 年に東北地区の最初の電子計算機として、東北大学電気通信研究所において完成されたパラメトロン式計算機の名前で SENAC-1(SENdai Automatic Computer-1)からとって命名された。

[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
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」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 C401-8 システムは、Linux OS 搭載の x86 ホストに、16 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、これまでの NEC SX シリーズ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。

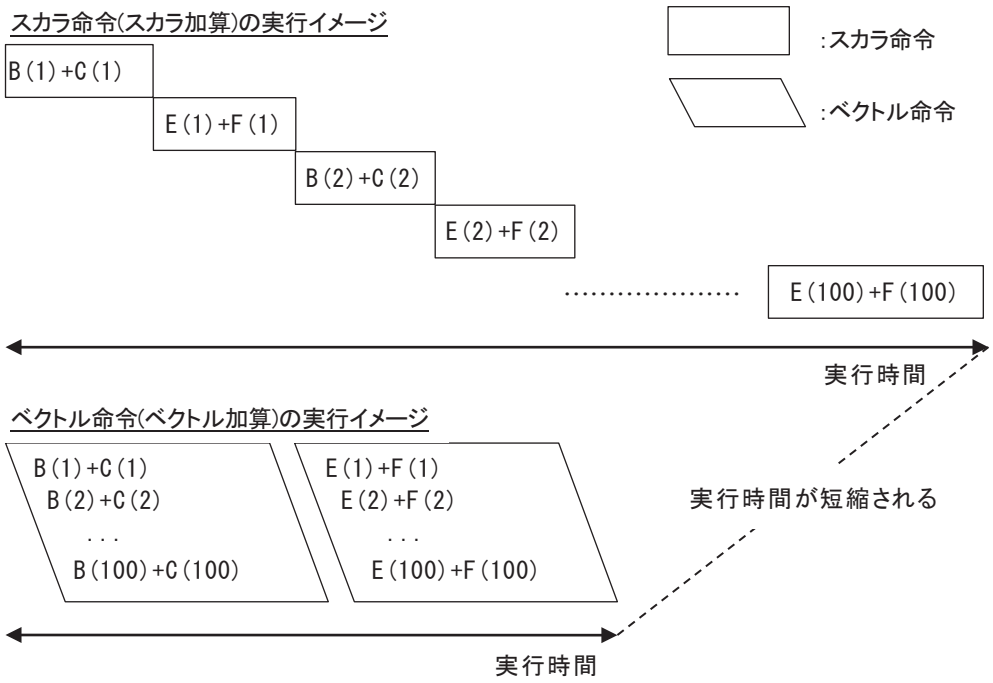


図-1　加算の実行イメージ

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=2)、INTEGER(KIND=4)、INTEGER(KIND=8)、REAL(KIND=2)、REAL(KIND=4)、REAL(KIND=8)、COMPLEX(KIND=2)、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
```

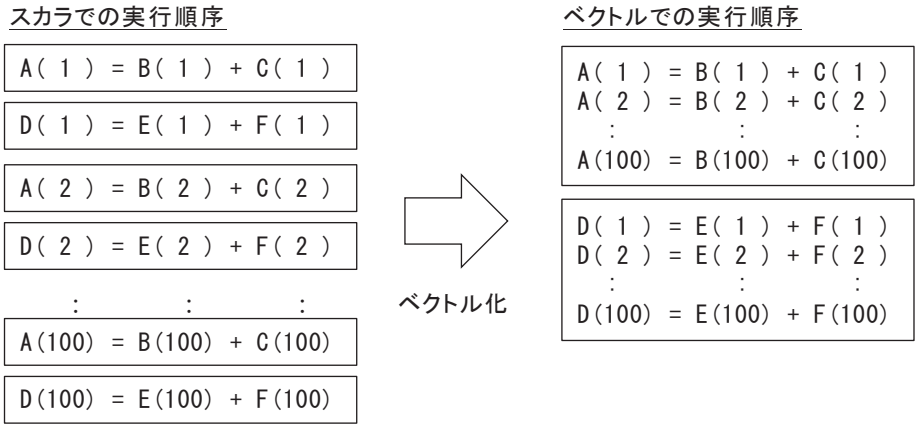


図-2　文の実行順序

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。

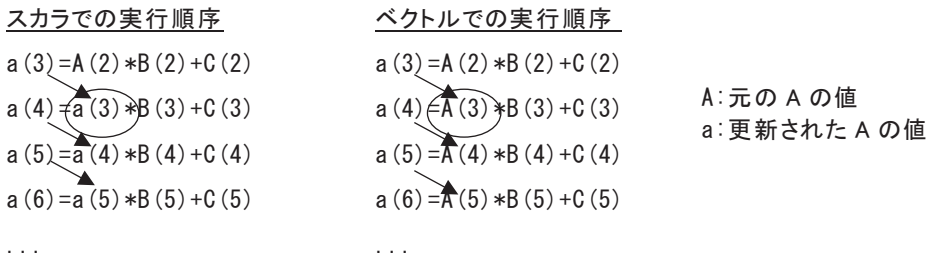


図-3　ベクトル化を阻害するデータの依存関係

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。

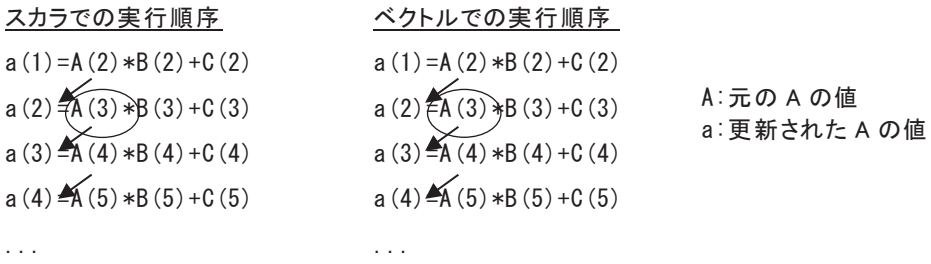


図-4　ベクトル化可能なデータの依存関係

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。

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。

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 (5.0.2) for Vector Engine   Fri Sep  8 13:25:29 2023
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 (5.0.2) for Vector Engine   Fri Sep  8 15:29:22 2023
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
```

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
- ベクトル融合積和演算(fused multiply-add, FMA)が使用された場合、途中の乗算の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。

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。

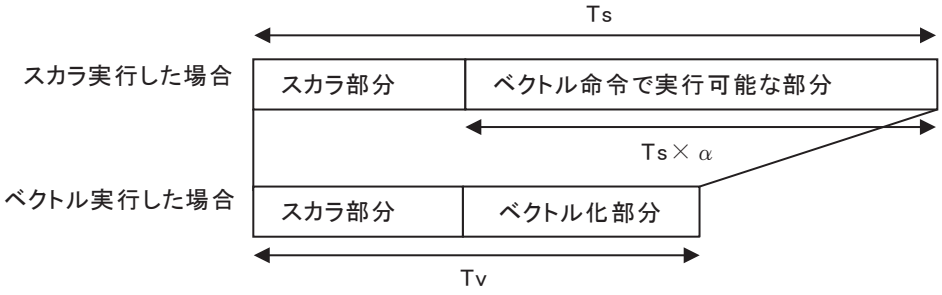


図-5 ベクトル化率

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-20 にプログラム実行解析情報の出力例を示します。

例-20

```
          ******  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
LD L3 Hit Element Ratio (%)         :             89.079104 L3キャッシュヒット率
VLD LLC Hit Element Ratio (%)       :             58.115252 ベクトル要素LLCヒット率
FMA Element Count                   :      214323200000    FMA 命令実行要素数
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
Non Swappable Memory Size Used (MB) :            128.000000 Swap outできないメモリ最大量
```

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。

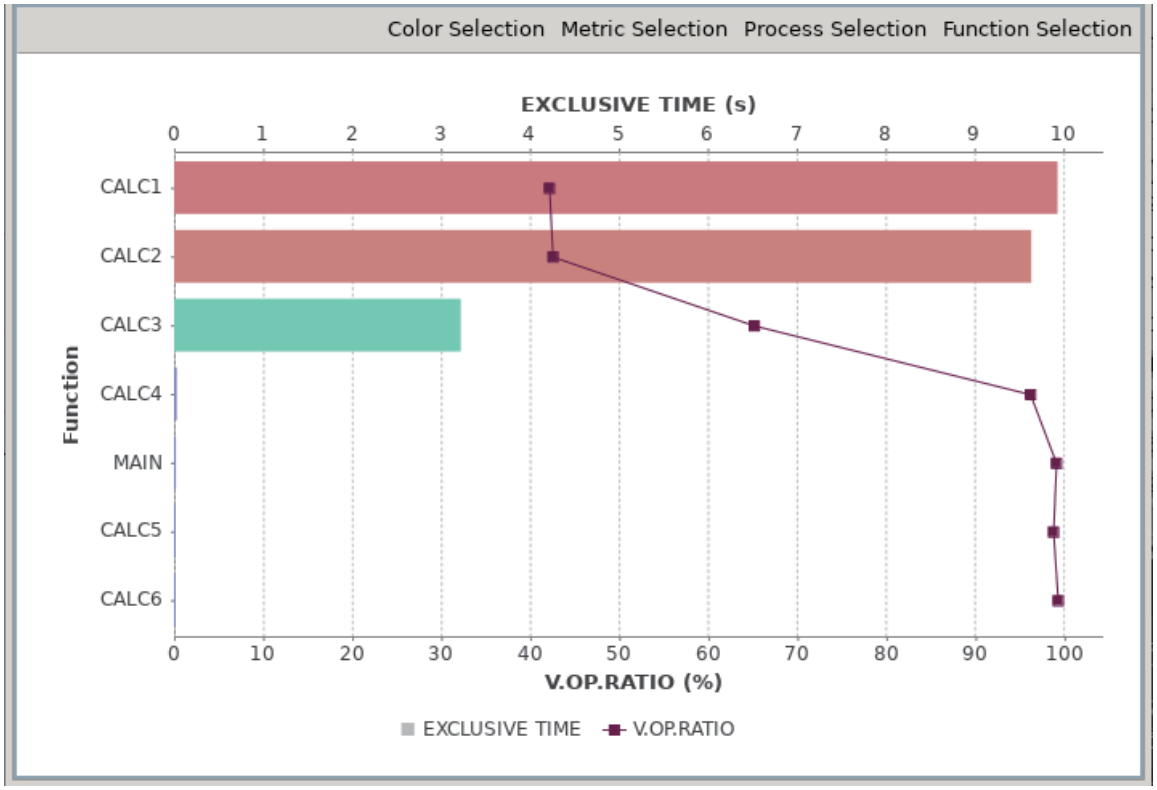
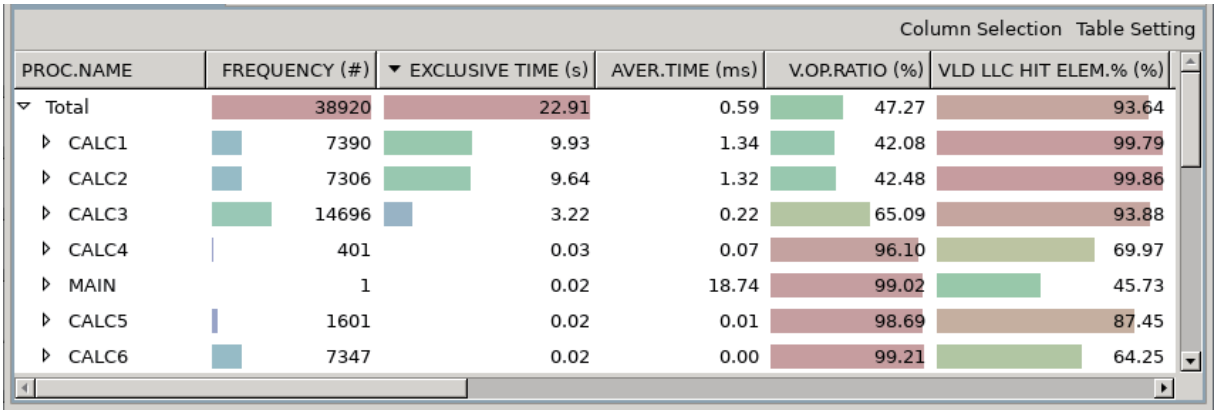


図-6 Function Metric Chart

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」[2] 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://sxauroratsubasa.sakura.ne.jp/NEC_Aurora_Forum

[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
https://sxauroratsubasa.sakura.ne.jp/Documentation(Japanese)

[3] NEC Ftrace Viewer ユーザーズガイド 日本電気株式会社
https://sxauroratsubasa.sakura.ne.jp/Documentation(Japanese)

［大規模科学計算システム］

SX-Aurora TSUBASA でのプログラミング（並列化編）

— 共有並列化と分散並列化 —

林 康晴　早坂 武
日本電気株式会社

サイバーサイエンスセンターのスーパーコンピュータ AOBA-S の運用が開始されました。AOBA-S に採用されている SX-Aurora TSUBASA は、第三世代の Vector Engine(VE30A)カードを 8 枚搭載した Linux サーバー(Vector Host)が、InfiniBand の Fat-Tree ネットワークにより接続されたシステムです。各 VE カードは、16 個の CPU コアを搭載しています。SX-Aurora TSUBASA のハードウェア性能を十分に引き出すには、プログラムの並列化により、複数の CPU コアを有効に活用する必要があります。SX-Aurora TSUBASA は、並列処理環境としてプログラミング言語 Fortran、C、および C++、並びに 通信ライブラリ MPI を用意しています。本稿は、第 1 章で並列処理の基本事項をご説明した後、第 2 章では NEC Fortran コンパイラ（以後、単にコンパイラと記します）による共有並列化、第 3 章では MPI ライブラリ NEC MPI による分散並列化を中心にご紹介します。

1. 並列処理

並列処理とは、ひとつの仕事を複数の小さな仕事に分割し、それらの仕事を複数同時に実行することです。例えば、2 重ループを 4 つの仕事に分割し、4 個の CPU コア上で並列実行すると、図 1.1 のようになります。この場合、外側の do j のループの 100 回の繰返しを 4 等分し、各 CPU コア上で並列に実行しています。

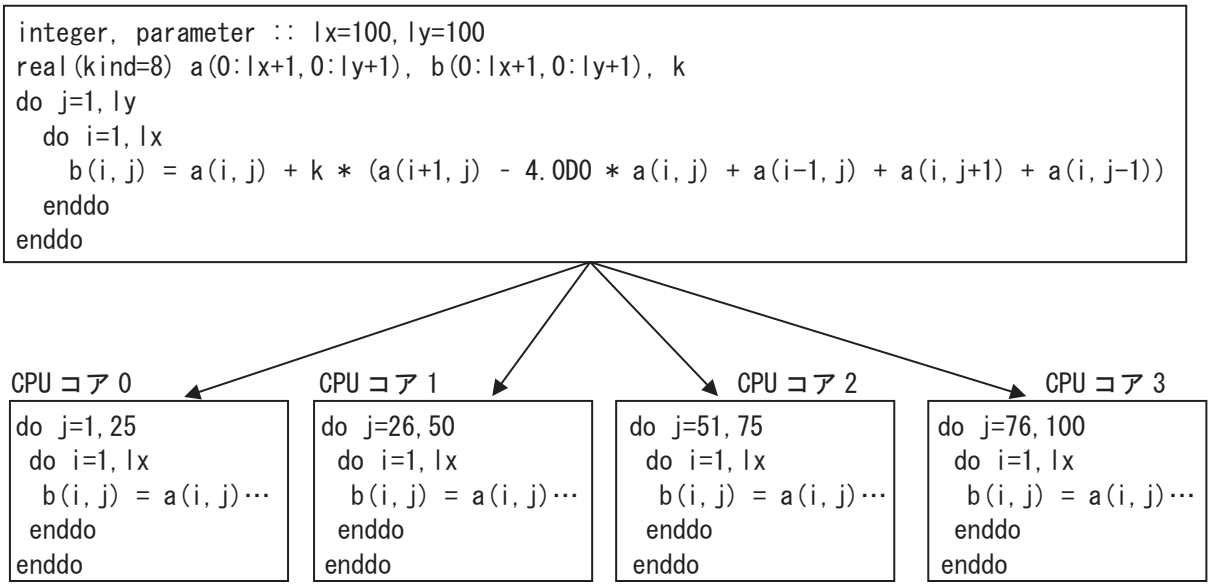


図 1.1　2 重ループの並列処理

1.1　並列化可能な条件

並列実行中、複数の仕事は、一般には実行タイミングの調整（同期）なしで、それぞれ独立に実行されます。そのため、異なる仕事内の計算の間には定まった実行順序はなく、一般には実行毎に異なる順序で実行されます。したがって、ループが並列実行可能であるためには、ループの繰返しをどのような順序で実行しても実行結果が変わらないことが必要です。例えば、図 1.2 の 2 つのループ(a)、(b)は、いずれもループ中で確定される各配列要素 a(i) および a(i,1), (i=2,3,…,n) が、各ループ(a)、(b)の全ての繰返

し中でそれぞれ 1 度しか出現しません。そのため、ループの繰返しをどのような順序で実行しても結果は同じになるので、並列実行可能です。一方、図 1.3 の(a)、(b)の場合、いずれもループの k 回目の繰返しで値が確定される配列要素 a(k)が、(a)の場合 k+1 回目の繰返しで、(b)の場合 k-1 回目の繰返しでそれぞれ引用されます。そのため、ループの繰返しの実行順序に依存して、確定前の値を引用するか、確定後の値を引用するかが変わってしまい、並列化することはできません。図 1.3 の(c)の場合も、ループの実行終了時に変数 ifound に残る値が、ループの繰返しの実行順序に依存して変わってしまうので、並列化することはできません。このように、ループを並列化するためには、ひとつの繰返しで確定したデータ要素は、他の繰返しでは確定も引用もしないようにループを記述する必要があります。図 1.3 の(d)のように、ループ外への飛越しを含むループも、ループの繰返しの実行順序に依存して飛越しのタイミングが変わってしまうので、並列化することはできません。

図 1.4 の 2 つのループ(a)、(b)は、それぞれスカラ変数 t、s をループの全ての繰返しで確定するので、図 1.3 の条件に該当し、一見並列化できないように思えます。しかし(a)の場合、変数 t は、ループの繰返し毎に新たに確定した値を、その繰返しの中だけで引用しています。そのため、ループ実行後に変数 t の値を引用しないのであれば、仕事毎に作業変数を確保して、並列実行中は、変数 t の代わりにその作業変数を参照すれば並列化可能です。また(b)のように、同一の変数に同一の演算を繰返し適用するパターンは集計計算と呼ばれます。集計計算の場合も、やはり仕事毎に作業変数を確保して、並列実行中はその作業変数に各仕事の局所的な計算結果を格納し、並列実行終了時に全ての仕事の計算結果を集計することにより並列化できます。

多重ループの場合、並列化できるかどうかはループ毎に判断します。図 1.5 の内側の do i のループは並列化可能ですが、外側の do j のループは、左辺の a(i,j)と右辺の a(i,j-1)との間に図 1.3(a)と同様の依存関係があるので並列化できません。

```
integer, parameter :: n=100
real(kind=8), dimension(n) :: a, b
do i=2,n
  a(i) = b(i) + b(i-1)
enddo
```

(a)

```
integer, parameter :: n=100
real(kind=8), dimension(n,n) :: a
do i=2,n
   a(i,1) = a(i,2) + a(i-1,2)
enddo
```

(b)

図 1.2　並列化できるループ

```
integer, parameter :: n=100
real(kind=8), dimension(n) :: a
do i=2,n
  a(i) = a(i) + a(i-1)
enddo
```

(a) 依存

```
integer, parameter :: n=100
real(kind=8), dimension(n) :: a
do i=1,n-1
   a(i) = a(i) + a(i+1)
enddo
```

(b) 逆依存

```
integer, parameter :: n=100
real(kind=8), dimension(n) :: a
integer ifound
do i=1,n
  if(a(i).ge.0) ifound = i
enddo
```

(c) 出力依存

```
integer, parameter :: n=100
real(kind=8), dimension(n) :: a
do i=1,n
  if(a(i).ge.0) goto 100
enddo
100 continue
```

(d) 制御依存

図 1.3　並列化できないループ

```
integer, parameter :: n=100
real(kind=8) :: a(n), b(n), t
do i=1,n-1
  t = a(i) + a(i+1)
  b(i) = t
enddo
```

(a) 繰返し内作業変数

```
integer, parameter :: n=100
real(kind=8) :: a(n), s
do i=1,n
  s = s + a(i)
enddo
```

(b) 集計計算

図 1.4　作業変数の使用により並列化できるループ

```
integer, parameter :: n=100
real(kind=8), dimension(n,n) :: a
do j=2,n
  do i=1,n
    a(i,j) = a(i,j-1) + a(i,j)
  enddo
enddo
```

図 1.5　内側ループだけ並列化できる多重ループ

1.2 並列化の効果

ひとつのプログラムを N 個の CPU コア上で並列実行した場合、最大 N 倍のスピードアップ（実行時間の短縮）が期待できます。このスピードアップを最大にするには、まず、どれだけ多くの部分を並列化できたか（並列化率）が重要です。これは、図 1.6 のように逐次実行時間の 10 %分の仕事が並列化できない場合、残りの仕事をどれだけ多くの小さな仕事に分割して並列化しても、スピードアップは 10 倍未満になってしまうことからも分かります。

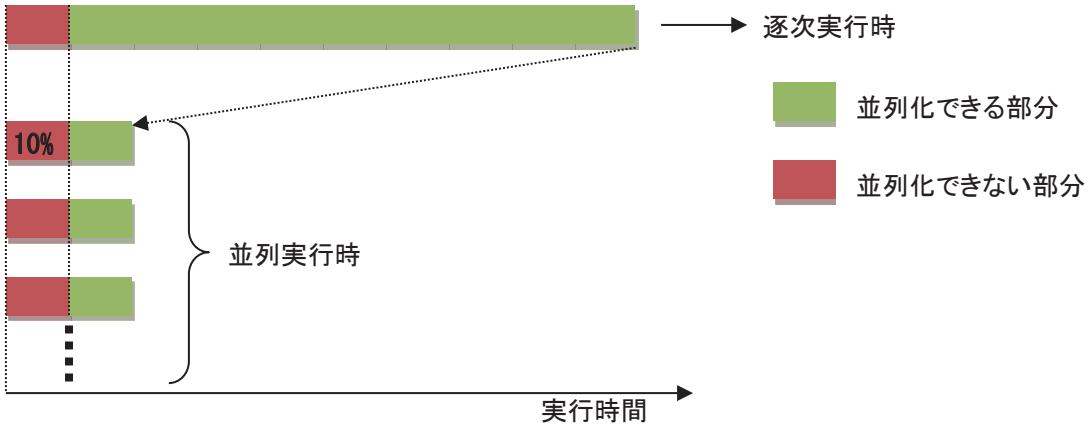


図 1.6　並列化率とスピードアップ（並列化率 90 %・スピードアップは 10 倍未満）

また、仕事を複数の小さな仕事に分割するための処理や、仕事間のデータのやり取り（通信）および同期といった逐次実行時には必要なかった処理（オーバヘッド）をできるだけ小さくすることも重要です。これは、図 1.7 のように逐次実行時間の 10 %分のオーバヘッドが並列化により発生すると、スピードアップはやはり 10 倍未満になってしまうことからも分かります。特に、実行時間（粒度）の小さなループを並列化するような場合、オーバヘッドの方が大きすぎ、並列化によりかえって遅くなる場合もあります。オーバヘ

ッドを削減するには、できるだけ外側のループで並列化する、といった方法により、各仕事の粒度をできるだけ大きくすると効果的です。

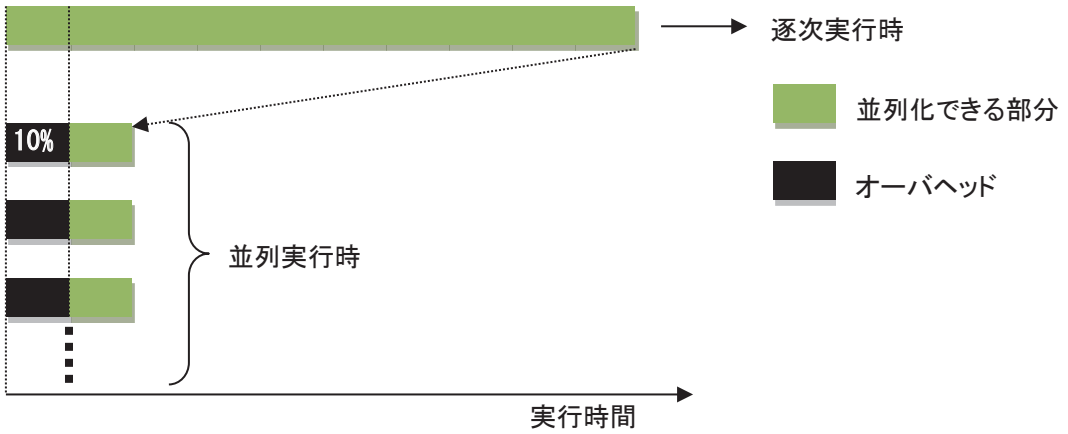


図 1.7　オーバヘッドとスピードアップ(オーバヘッド 10 %・スピードアップは 10 倍未満)

さらに、各仕事の実行時間のバランス(負荷バランス)をできるだけ均等化することも重要です。これは、図 1.8 のように複数の仕事のうちひとつの仕事の実行に逐次実行時の 10 %分の時間がかかってしまったとすると、残りの仕事をどれだけ高速化しても、スピードアップは 10 倍以下になってしまうことからも分かります。

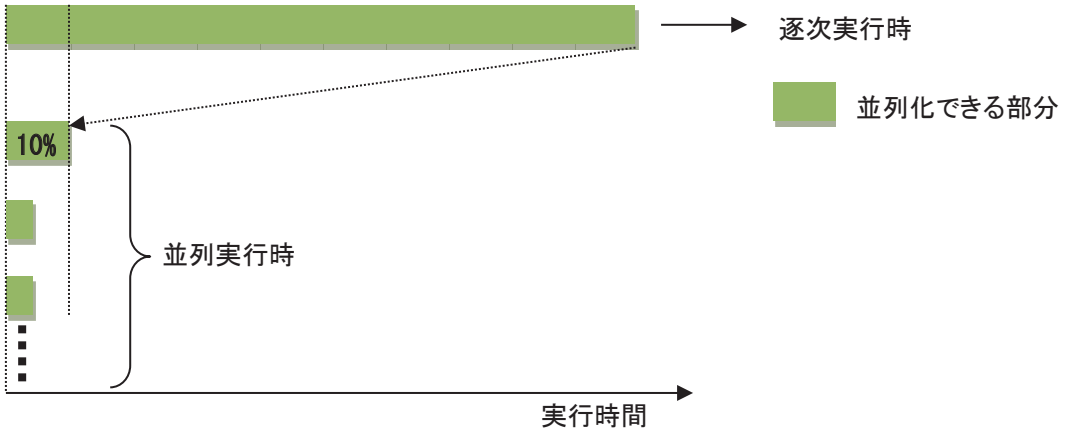


図 1.8　負荷バランスとスピードアップ(スピードアップは 10 倍以下)

このように、並列プログラムにおいて高いスピードアップを達成するには、高い並列化率・低いオーバヘッド・均等な負荷バランスの 3 点が重要です。

1.3　経過時間と CPU 時間

並列処理は、ひとつの仕事を複数の小さな仕事に分割して、複数の CPU コア上で同時に実行することにより、仕事の実行時間(経過時間)を減らします。一方、分割した全ての仕事を実行するための CPU コアの処理時間の総計(CPU 時間)を減らすことはできません。実際には、ひとつの仕事を複数の小さな仕事に分割するための処理や仕事間の同期などのオーバヘッドも発生するので、CPU 時間は並列化によりむしろ増加することに注意してください。

1.4 経過時間とチューニング

並列化の主たる目的であるプログラム実行の経過時間短縮のためには、アルゴリズムの工夫、高速なライブラリの使用、およびベクトル化の促進などにより、逐次プログラムとしての性能を並列化の前に十分チューニングしておくことが重要です。並列化によるスピードアップは、使用する CPU コアの個数が上限となりますが、逐次プログラムのチューニングによる高速化は、場合により数十～数百倍に達することもあり、さらに CPU 時間も削減できるからです。

2. 共有並列化

一般に、コンピュータのプログラムは、プロセスにより実行されます。実行中のプロセスは、そのメモリ空間にプログラムのデータを格納しており、現在の実行状態をもっています。プロセスは、複数のスレッドから構成することもできます。ひとつのプロセス中の複数のスレッドは、それぞれ独自の実行状態をもち、別々の仕事を実行できますが、プロセスのメモリ空間は全てのスレッドが共有します。

共有並列化とは、ひとつのプロセス中の複数のスレッドによる並列処理のことです。AOBA-S の各 VE カード内では、主記憶装置を共有する 16 個の CPU コア上で実行されるスレッドに仕事を割り当て、共有並列化を行うことができます。例として、図 1.1 の 2 重ループの共有並列化を考えます。第 1 章と同様に、外側の do j のループを 4 つの仕事に分割して別々のスレッドに割り当てます。この場合、データ領域に関しては、図 2.1 のように、2 次元配列 a、b を do j のループに対応する 2 次元目で分割し、分割された各部分領域の処理を各スレッドが担当することになります。ここで、例えば j=25 の繰返しを担当するスレッド 0 は、スレッド 1 の担当領域である配列要素 a(i,26), (i=1,2,…,lx) の値を引用する、といったように、配列の分割境界部分の処理では、自スレッドの担当領域外の配列要素を参照する必要があります。しかし、各スレッドは、メモリ上に配置された 2 次元配列 a、b の全ての領域を直接参照できるので、特に問題はありません。一方、DO 変数 i、j については注意が必要です。並列実行中、各スレッドは変数 i、j をそれぞれ独自のタイミングで参照します。そのため、配列 a、b と同様に、変数 i、j の領域を全てのスレッドが共有すると、あるスレッドが確定した変数 i、j の値を別のスレッドが引用してしまう可能性があり、実行が正しく行えません。従って、並列実行中に各スレッドが値を確定する変数は、図 2.2 のようにそれぞれのスレッドが固有の作業領域を割り付けて参照する必要があります。2 次元配列 a、b のような全てのスレッドがメモリ領域を共有するデータを SHARED データ、DO 変数 i、j のような各スレッドが専用のメモリ領域を割り付けるデータを PRIVATE データと呼びます。

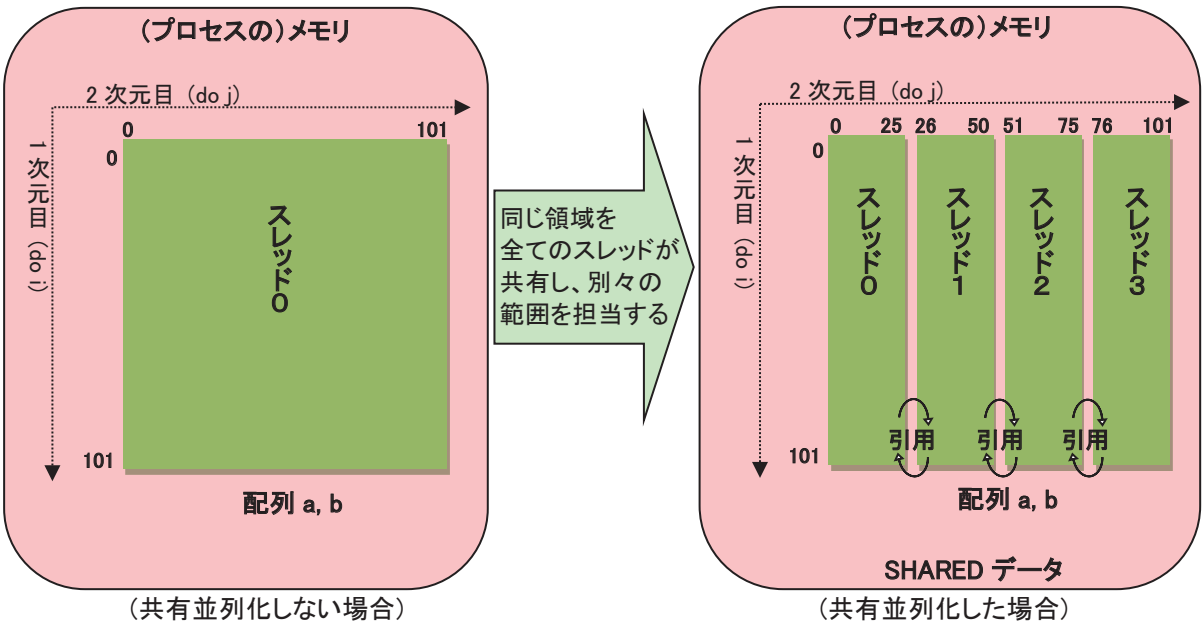


図 2.1　各スレッドが担当する配列 a、b の領域

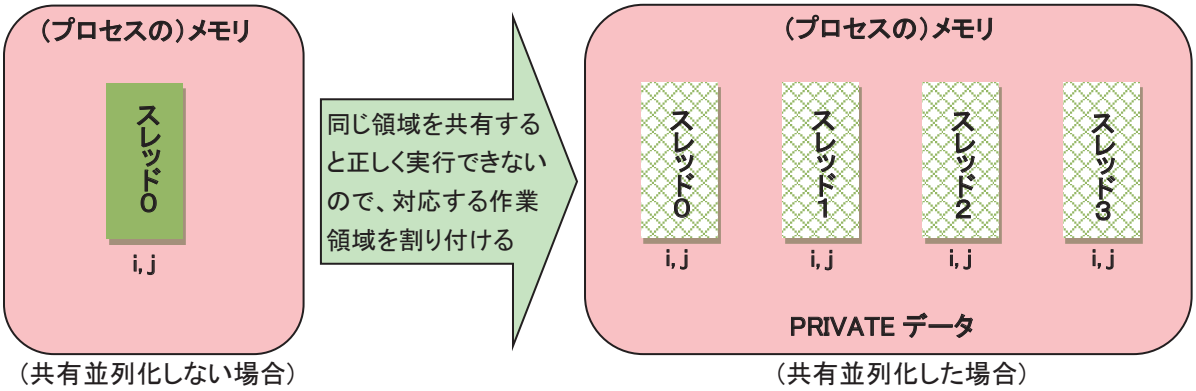


図 2.2　各スレッド専用の DO 変数 i、j 用の作業領域

2.1　自動並列化

並列化を行う場合、並列実行しても結果が不正にならないことを保証するために、通常はデータの依存関係の解析を行い、細心の注意を払ってプログラムの変形や指示文の挿入を行う必要があります。しかし、コンパイラの自動並列化機能を使用すると、コンパイラオプション-mparallel を翻訳時に指定するだけで、それらの作業をコンパイラが自動的に行います。

コンパイラは、まずプログラムを解析して並列実行可能なループや文の集まりを抽出し、次にそれらを複数の仕事に分割してスレッドに割り当て、経過時間を短縮します。また、SHARED データ・PRIVATE データの判定も自動的に行います。コンパイラの自動並列化対象は、図 2.3 のとおりです。

対象となる構文	DO ループ、配列式
対象ループ中に書ける文	代入文、IF 構文、GOTO 文、CONTINUE 文、CALL 文、CASE 構文
対象となる演算	四則演算、べき乗演算、論理演算、関係演算、型変換、組込み関数

図 2.3 コンパイラの自動並列化対象

2.2　コンパイラの最適化

ここでは、共有並列化の効果を高めるためにコンパイラが行う最適化をいくつかご紹介します。

2.2.1　ループ変形

ループ融合・ループ一重化などのループ変形による最適化が可能な場合、最適化後に共有並列化を行い、共有並列実行時の仕事の粒度を最大化します。図 2.4 にループ融合の例を示します。

```
integer, parameter :: n=100
real(kind=8), dimension(n) :: a, b, c, d
do i=1,n
  c(i) = a(i) + sqrt(b(i))
enddo
do i=1,n
  d(i) = a(i) + sqrt(b(i))
enddo
```

(ループ融合前)

```
do i=1,n
  c(i) = a(i) + sqrt(b(i))
  d(i) = a(i) + sqrt(b(i))
enddo
```

(ループ融合後)

図 2.4　ループ融合の例

2.2.2 条件並列化

並列化できるかどうか、または並列化により高速化できるかどうかが翻訳時に判断できない場合、実行時に依存関係やループ長を調べて、共有並列化するかどうかを選択できるよう条件並列化を行います。図 2.5 では、条件並列化後の IF 構文の論理式 nx*ny > n によって、共有並列化により高速化するのに十分なループ長があるかどうかを判定しています。この際、ループ中の各演算の演算コストに基づいて、共有並列化の効果が期待できる値(n)を自動的に算出し、条件並列化を行います。また、論理式 id-ic==0 .or. abs(id-ic)>=nx によって、ループ中で確定される配列要素 y(ic+i)と y(id+i)の領域に重なりがなく、並列化可能であるかどうかを判定しています。

```
do i = 1,nx
  aa = a
  bb = b
  do j = 1,ny
    aa = aa + x(i+1) * g(j-1)
    bb = bb + x(i-1) * g(j-1)
  enddo
  y(ic+i) = -aa
  y(id+i) = -bb
enddo
```

（条件並列化前）

```
if((nx*ny > n).and.(id-ic==0 .or. abs(id-ic)>=nx))then
  並列コード
else
  非並列コード
endif
```

（条件並列化後）

図 2.5　条件並列化の例

2.3 自動並列化促進のためのプログラミング

コンパイラオプション-mparallel の指定だけでは自動並列化できない場合でも、コンパイラ指示行の挿入やプログラムの修正により自動並列化できる場合もあります。ここでは、主な並列化用指示行とプログラムの修正例をご紹介します。

2.3.1 並列化指示行

コンパイラ指示行の書式は、図 2.6 のとおりです。

Fortran の場合：

```
!NEC$ コンパイラ指示オプション
```

C・C++の場合：

```
#pragma _NEC コンパイラ指示オプション
```

図 2.6　コンパイラ指示行の書式

自動並列化のために用意されている主なコンパイラ指示オプションは、次のとおりです。

- concurrent [schedule(種別[, 繰返し回数])] / noconcurrent
  直後のループの自動並列化を許可する / しないを指定します。コンパイラ指示オプション concurrent には、省略可能な schedule 指示句を指定して、ループの繰返しをどのようにスレッドに割り当てるかを指定できます。種別として指定できるのは、static、dynamic、または runtime です。static

は、ループの繰返しを、スレッド番号の昇順にラウンドロビン方式で割り当てます。dynamic は、ループの繰返しを、その時点で仕事が割り当たっていないスレッドに割り当てます。runtime は、実行時に環境変数 OMP_SCHEDULE で指定された種別に従います。static および dynamic には、一度に割り当てるループの繰返し回数を指定できます。コンパイラ指示オプション noconcurrent は、粒度が小さく共有並列化するとかえって性能が低下するループに指定します。

- inner / noinner
  内側ループまたは一重ループを自動並列化の対象とする / しないを指定します。多重ループの内側ループは、既定値では自動並列化の対象にならないので、共有並列化したい場合、コンパイラ指示オプション inner を指定します。一重ループは、共有並列化の効果が翻訳時に不明の場合、自動並列化の対象になりませんが、コンパイラ指示オプション inner を指定することにより自動並列化の対象とすることができます。

- select_concurrent
  多重ループの内、直後のループを優先して自動並列化します。

- nosync
  ループ中で参照される配列に依存がないことを指定します。依存関係が翻訳時には分からないので自動並列化できないときでも、依存がないことを利用者が分かっている場合、コンパイラ指示オプション nosync によりそれをコンパイラに教えてやることによって、自動並列化できる場合があります。

- cncall
  Fortran の組込み関数以外の手続引用を含むループは、既定値では自動並列化の対象になりませんが、並列化しても問題ないことを利用者が分かっている場合、コンパイラ指示オプション cncall を指定すると自動並列化の対象とすることができます。自動並列化されたループ中で引用される手続の仮引数は、結合する実引数が SHARED データの場合、原則として SHARED データとなるので、そのような仮引数を手続中で確定した結果、図 1.3 のような依存関係が発生すると結果が不正となることに注意してください。

2.3.2 並列化のためのソース変形

図 2.7(a)の 2 重ループは、外側の do j のループには依存があるので並列化できませんが、コンパイラ指示オプション inner を内側の do i のループに指定すると、内側ループで自動並列化されます。ただし、内側ループを並列化すると、並列化のオーバヘッドが外側ループの繰返し回数分発生してしまいます。このような場合、図 2.7(b)のように内側ループと外側ループを利用者が交換すると、外側ループで自動並列化され並列化のオーバヘッドを削減できます。

```
integer, parameter :: n=10000,m=10000
real(kind=8), dimension(n,m) :: a
do j=1,m-1
   do i=1,n
      a(i,j) = a(i,j+1) + a(i,j)
   enddo
enddo
```

(a) ループ交換前

```
do i=1,n
 do j=1,m-1
   a(i,j) = a(i,j+1) + a(i,j)
 enddo
enddo
```

(b) ループ交換後

図 2.7　ループ交換の例

図 2.8(a)のループネストは、仮引数 w に関するデータ依存により、外側の do j のループを自動並列化できません。このような場合、図 2.8(b)のように仮配列引数 w を次元拡張し、do j のループの繰返し毎に異なる要素を参照するように修正すれば、自動並列化可能となります。この際、仮引数 w に対応する実引数にも同様の修正が必要となることに注意してください。また、手続 sub の引用元が、仮引数 w に対応する実引数を参照しておらず、配列 w が単なる作業変数として利用されている場合は、図 2.8(c)のように仮引数 w を手続 sub の局所変数に変更すると、ループネスト中ではコンパイラにより PRIVATE データとして割り付けられるので、自動並列化できます。

```
subroutine sub(a,b,w)
parameter(n=100,m=100)
real(kind=8) a(n,m),b(n,m),w(0:n)
do j=1,m
  do i=1,n
    w(i) = a(i,j) / dble(n)
  enddo
  do i=1,n
    b(i,j) = b(i,j) + (w(i-1)+w(i))/2.0
  enddo
enddo
```

(a) 仮引数 w に関する依存

```
subroutine sub(a,b,w)
parameter(n=100,m=100)
real(kind=8) a(n,m),b(n,m),w(0:n,m)
do j=1,m
 do i=1,n
   w(i,j) = a(i,j) / dble(n)
 enddo
 do i=1,n
   b(i,j) = b(i,j)+(w(i-1,j)+w(i,j))/2.0
 enddo
enddo
```

(b) 次元拡張後

```
subroutine sub(a,b)
parameter(n=100,m=100)
real(kind=8) a(n,m),b(n,m),w(0:n)
do j=1,m
  do i=1,n
    w(i) = a(i,j) / dble(n)
  enddo
  do i=1,n
    b(i,j) = b(i,j) + (w(i-1)+w(i))/2.0
  enddo
enddo
```

(c) 局所変数化後

図 2.8　仮引数の修正により自動並列化可能となる例

2.5 OpenMP による共有並列化

コンパイラは、OpenMP による手動並列化も用意しています。OpenMP は、the OpenMP Architecture Review Board (ARB)[1] によって策定された共有並列化のためのオープンな API であり、共有並列化のデファクトスタンダードとして広く使用されています。OpenMP では、主として、ソースプログラムに OpenMP ディレクティブを挿入することによって共有並列化を行います。利用者は、OpenMP ディレクティブに必要に応じて指示句を追加し、並列化方法を詳細に制御できます。OpenMP ディレクティブの書式は、図 2.9 のとおりです。

Fortran の場合:

```
!$OMP OpenMP ディレクティブ
```

C・C++の場合:

```
#pragma omp OpenMP ディレクティブ
```

図 2.9　OpenMP ディレクティブの書式

SX-Aurora TSUBASA で OpenMP を使用する場合、コンパイラオプション-fopenmp を翻訳時に指定してください。プログラム実行の際は、1 つのプロセスが利用するスレッド数を環境変数 VE_OMP_NUM_THREADS で指定してください。

2.5.1 OpenMP 使用時の注意点

自動並列化の場合、コンパイラがプログラムを解析し並列ループ間の不必要な同期を自動的に抑制します。一方 OpenMP による共有並列化時には、nowait 指示句を指定しない限り並列ループの直後に毎回同期が生成されます。同期が必要ない場合には、nowait 指示句を指定すると共有並列化のオーバヘッドを削減できます。

また自動並列化の場合、図 1.4 のような作業変数や集計変数は、コンパイラによって自動的に認識され適切に処理されます。一方 OpenMP による共有並列化時には、DO 変数以外の作業変数および集計変数をそれぞれ private 指示句および reduction 指示句中に指定する必要があります。

OpenMP の parallel do 指示文をループに指定して共有並列化した例を図 2.10 に示します。繰返し内作業変数 t を private 指示句中に、集計演算子+ および集計変数 s を reduction 指示句中に指定していることに注意してください。なお、DO 変数 i は、この場合既定値で PRIVATE データとなるので、private 指示句を指定する必要はありません。

```
integer, parameter :: n=100
real(kind=8) :: a(n), t, s
!$OMP parallel do private(t) reduction(+: s)
do i=1,n-1
  t = a(i) + a(i+1)
  s = s + t
enddo
```

図 2.10　OpenMP による並列化例

2.6 性能解析

2.6.1 プログラム実行性能情報（PROGINF）

逐次（ベクトル）プログラムの場合と同様に、共有並列プログラムの場合も、プログラム実行性能情報（PROGINF）が使用できます。PROGINF は、環境変数 VE_PROGINF の値を YES または DETAIL に設定してプログラムを実行することによって採取できます。図 2.11 に、環境変数 VE_PROGINF の値を DETAIL に設定して共有並列実行した場合の出力例を示します。ここで図中の※は、共有並列実行時に固有の情報です。 Max Active Threads の値により、同時に実行したスレッドの個数の最大値を確認できます。また、図中◇は、VE30A から追加された情報であり、レベル 3 キャッシュの利用率情報を出力します。

Real Time (sec)	:	18.914921	
User Time (sec)	:	297.227732	
Vector Time (sec)	:	204.714487	
Inst. Count	:	225908581946	
V. Inst. Count	:	60747556381	
V. Element Count	:	15380715794317	
V. Load Element Count	:	4512736316356	
FLOP Count	:	6217859387690	
MOPS	:	58868.836694	
MOPS (Real)	:	925048.247358	
MFLOPS	:	20919.512947	
MFLOPS (Real)	:	328723.308860	
A. V. Length	:	253.190691	
V. Op. Ratio (%)	:	99.056085	
L1 Cache Miss (sec)	:	14.285157	
CPU Port Conf. (sec)	:	0.000000	
V. Arith. Exec. (sec)	:	132.062536	
V. Load Exec. (sec)	:	64.239533	
LD L3 Hit Element Ratio (%)	:	39.529025	◇
VLD LLC Hit Element Ratio (%)	:	42.832553	
FMA Element Count	:	1951573996260	
Power Throttling (sec)	:	0.000000	
Thermal Throttling (sec)	:	0.000000	
Max Active Threads	:	16	※
Available CPU Cores	:	16	※
Average CPU Cores Used	:	15.713929	※
Memory Size Used (MB)	:	2778.000000	
Non Swappable Memory Size Used (MB)	:	102.000000	

図 2.11　共有並列実行時の PROGINF

2.6.2 性能解析機能(FTRACE 機能)

逐次(ベクトル)プログラムの場合と同様に、共有並列プログラムの場合も、FTRACE 機能によって手続単位又は利用者が指定した区間単位の詳細な性能情報が取得できます。

基本的な使用方法は逐次プログラムの場合と同様ですが、コンパイラは、共有並列化する各ループネストを切り出して、それらを独立した手続に変形します。そのため共有並列化された各ループネストは、FTRACE の表示上は独立したひとつの手続として表示されることに注意してください。切りだされた各ループネストは、ソースプログラム中の出現順に、元の手続名に加えて_$1、_$2、・・・とサフィックスが付いた名前の手続として表示されます。図 2.12 に、共有並列実行時の FTRACE の表示例を示します。_$1 のついた手続 JACOBI$1 は、手続 JACOBI の最初に出現する共有並列化されたループネストの性能情報です。-thread0、-thread1、・・・は、各スレッドの性能情報です。JACOBI のような$のついていない手続名は、手続 JACOBI 中の共有並列化されなかった部分の性能情報です。図中の「LD L3 HIT E.%」は VE30A から追加された情報であり、レベル 3 キャッシュの利用率情報を出力します。

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER. TIME [msec]	MOPS	MFLOPS	V. OP RATIO	AVER. V. LEN	VECTOR TIME	L1CACHE MISS	CPU PORT CONF	LD L3 HIT E. %	VLD LLC HIT E. %	PROC. NAME
32	246.090(100.0)	7690.305	71014.7	25266.6	99.18	253.2	204.711	14.283	0.000	39.53	42.83	JACOBI$1
2	15.382(6.3)	7691.243	72129.3	25664.6	99.18	253.2	13.167	0.856	0.000	39.34	41.86	-thread0
2	15.380(6.2)	7689.957	72142.2	25668.9	99.18	253.2	13.030	0.887	0.000	39.52	43.15	-thread1
2	15.380(6.2)	7689.916	72141.6	25669.0	99.18	253.2	13.111	0.890	0.000	39.50	43.01	-thread2
2	15.381(6.3)	7690.571	72136.3	25666.8	99.18	253.2	13.104	0.905	0.000	39.52	42.94	-thread3
	:											
2	0.000(0.0)	0.010	383.1	0.0	0.00	0.0	0.000	0.000	0.000	46.91	0.00	JACOBI

図 2.12　共有並列実行時の FTRACE

主に浮動小数点演算を実行する共有並列プログラムの性能を分析する場合、FTRACE 中の MFLOPS の値に着目してください。図 2.11 の場合、並列ループネスト JACOBI$1 は、4 個のスレッドそれぞれが約 26000MFLOPS の性能で実行したことが分かります。MFLOPS 値がスレッド間でほぼ均等となっているので、うまく共有並列化されていると判断できます。逆に図 2.13 のように、スレッド 0 およびスレッド 1 だけ MFLOPS 値が大きく、残りのスレッドが小さい場合は、負荷バランスが悪い、と判断できます。このようなプログラムは、コンパイラ指示オプション concurrent の schedule 指示句などを指定して負荷バランスを均等化すると、経過時間を短縮できる可能性があります。

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER. TIME [msec]	MOPS	MFLOPS	V. OP RATIO	AVER. V. LEN	VECTOR TIME	L1CACHE MISS	CPU PORT CONF	LD L3 HIT E. %	VLD LLC HIT E. %	PROC. NAME
32	676.916(100.0)	21153.626	11577.0	4082.8	98.30	253.2	81.652	4.896	0.000	41.05	43.12	JACOBI$1
2	42.309(6.3)	21154.469	90631.2	33179.7	99.24	253.2	38.506	2.139	0.000	42.71	44.64	-thread0
2	42.307(6.2)	21153.667	87815.0	32144.0	99.24	253.2	37.900	2.142	0.000	42.37	43.45	-thread1
2	42.307(6.2)	21153.285	490.9	0.0	73.98	254.0	0.400	0.036	0.000	0.01	0.01	-thread2
2	42.307(6.2)	21153.746	490.9	0.0	73.98	254.0	0.421	0.044	0.000	0.04	0.04	-thread3

図 2.13　負荷バランスの悪い共有並列実行時の FTRACE

3. 分散並列化

分散並列化とは、それぞれ独自のメモリ空間をもつ複数のプロセスによる並列処理のことです。SX-Aurora TSUBASA では、ひとつまたは複数の VE カードの CPU コア上で実行されるプロセスに仕事を割り当て、分散並列化を行うことができます。

例として、図 1.1 の 2 重ループの分散並列化を考えます。第 1 章と同様に、外側の do j のループを 4 つの仕事に分割して別々のプロセスに割り当てます。この場合、データ領域に関しては、図 3.1 のように、2 次元配列 a、b を do j のループに対応する 2 次元目で分割し、分割された各部分領域の処理を各プロセスが担当することになります(分割配置・1 次元分割)。ここで、例えば j=25 の繰返しを担当するプロセス 0 は、プロセス 1 の担当領域である配列要素 a(i,26), (i=1,2,…,lx)の値を引用する、といったように、配列の分割境界部分の処理では、自プロセスの担当領域外の配列要素を参照する必要があります。ところが各プロセスは、他のプロセスに割り付けられた配列の領域を直接参照できないので、通信を行って必要な配列要素の値を取得しなければなりません。このように分散並列化の場合、共有並列化と異なり、他のプロセスの担当範囲のデータを参照するために、必要に応じて通信が必要です。一方 DO 変数 i、j については、図 3.2 のように各プロセスがそれぞれ専用の領域を割り付ける(重複配置)と、並列実行中、各

プロセスは変数 i、j をそれぞれ独立に参照できるので通信の必要はありません。すなわち、分散並列実行時にどのような通信が必要となるかは、各データ要素をどのようにプロセスに割り付けるか（データマッピング）およびひとつの仕事をどのように複数の小さな仕事へと分割してプロセスに割り当てるか（計算マッピング）に依存します。このように分散並列化においては、データマッピング、計算マッピング、および通信の 3 点を全て考慮してプログラムする必要があります。

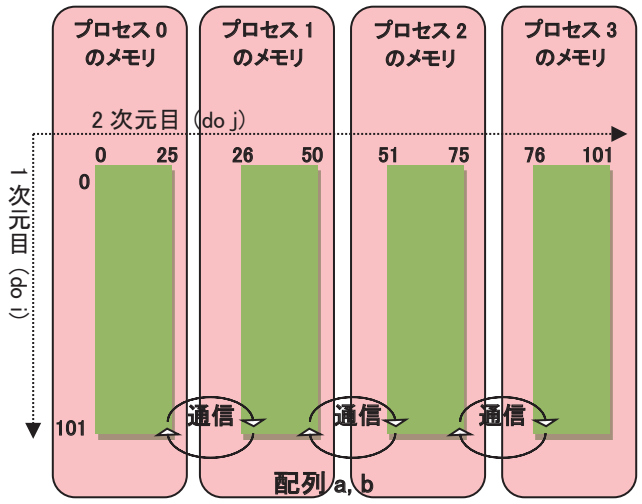


図 3.1　配列 a、b の分割配置（1 次元分割）

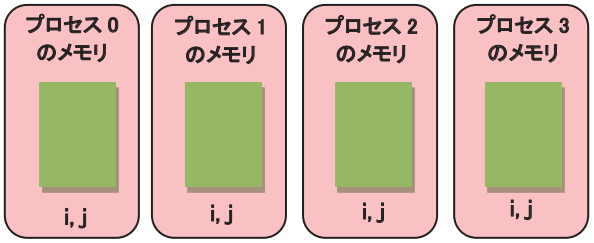


図 3.2　DO 変数 i、j の重複配置

3.1　データマッピングと性能

再び図 1.1 の 2 重ループの分散並列化を考えます。今度は図 3.3 のように、外側・内側のループをそれぞれ 2 つの仕事に分割してみます。この場合、データ領域に関しては、図 3.4 のように 2 次元配列 a、b を 1 次元目・2 次元目ともに分割し、分割された各部分領域の処理を各プロセスが担当することになります（分割配置・2 次元分割）。ここでも、例えば j=50 の繰返しを担当するプロセス 0 は、プロセス 2 の担当領域である配列要素 a(i,51), (i=1,2,…,50)の値を引用する、といったように、配列の分割境界部分の処理では、自プロセスの担当領域外の配列要素を参照するために通信が必要です。

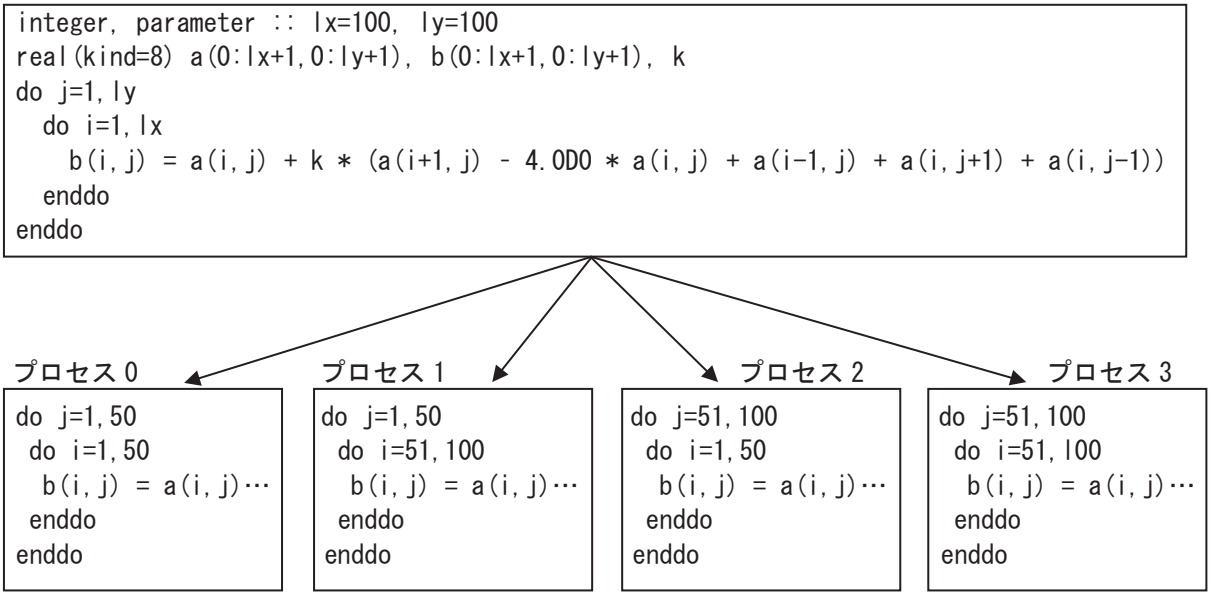


図 3.3　2 つのループをともに並列化する例

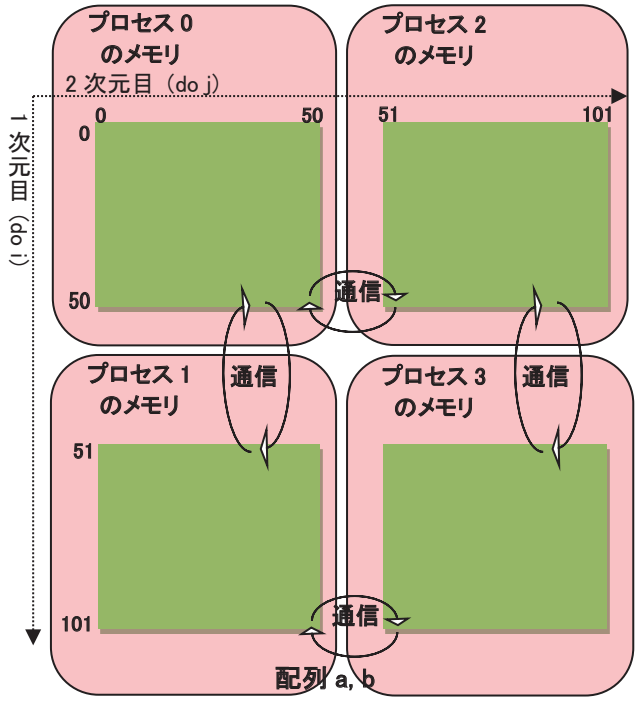


図 3.4　配列 a、b の分割配置(2 次元分割)

これを前節の外側ループだけによる分散並列化(1 次元分割)と比較してみます。各プロセスの通信に関しては、1 次元分割の場合、送受信回数はそれぞれ 2 回(2 次元目の上下方向)、送受信量はそれぞれ 1 次元目の寸法である lx*2 となります。一方 2 次元分割の場合、送受信回数はそれぞれ 4 回(1・2 次元目それぞれの上下方向)、送受信量はそれぞれ (lx/1 次元目分割数+ly/2 次元目分割数)*2 となります。従って、1 次元分割と比べると、2 次元分割の方が通信回数は多いですが、通信量はプロセスの個数が増えるほど少なくなります。1 回の通信時間は、典型的にはセットアップ時間+通信量/通信バンド幅と表せます。そのため、通信量が少なく(配列の寸法が小さく)セットアップ時間の影響が大きい場合には 1 次元分割の方が通信コストは少なく、逆に通信量が多く(配列の寸法が大きく)通信量の影響が大きい場合には 2 次元分割の方が通信コストは少なくなると考えられます。一方、各プロセスの演算性能を比較

すると、ベクトル長は内側の do i のループに対応する配列の 1 次元目の寸法に依存します。そのため、1 次元目をプロセス間に分割することによってベクトル長が短くなりすぎると、ベクトル演算の効率が低下します。従って、配列の寸法がそれほど大きくない場合、2 次元目だけを分割したほうが演算時間は短い可能性があります。このように、これら 2 つの分散並列化方法のどちらが経過時間を短縮できるかは、プロセスの個数や配列の寸法に依存して変わります。分散並列化時には、通信コストや並列化後の演算性能を考慮して、データマッピングを決定することが重要です。

3.2 Message Passing Interface (MPI)

3.2.1 MPI 概要

MPI は、Message Passing Interface Forum[2] によって策定された通信ライブラリのオープンな規格であり、分散並列プログラミングのデファクトスタンダードとして広く使用されています。MPI を使用した分散並列化では、図 3.1 や図 3.4 中の通信部分を、MPI 手続の引用によってプログラムすることになります。

分散並列化時に高いスピードアップを達成するには、通信コストの削減が特に重要です。プロセス内のメモリアクセスコストと比較して、プロセス間の通信コストは圧倒的に大きいからです。

SX-Aurora TSUBASA は、高速な通信環境を提供するため、MPI ライブラリ NEC MPI を用意しています。NEC MPI は、プロセスの VE カードへの配置方法や通信サイズに応じて、VE カード上の主記憶装置や InfiniBand のハードウェア機能を利用した最適な通信プロトコルを選択し、SX-Aurora TSUBASA の性能を最大限引き出しています。

3.2.2 MPI プログラミングの基本

MPI プログラミングでは、まず MPI 規格で規定されている各種定数などが定義されたヘッダファイル mpi.h をインクルード（C・C++の場合）、あるいは、システムモジュール mpi または mpi_f08 を引用します（Fortran の場合）。なお、Fortran の場合は、ヘッダファイル mpif.h をインクルードすることもできますが、mpif.h の利用は推奨されません。次に、手続 mpi_init または mpi_init_thread を引用して MPI の初期化を行い、手続 mpi_finalize を引用して MPI の使用を終了します。利用者は、MPI の初期化から使用終了までの間で、一対一通信、集団通信、または片側通信により通信を行うことができます。

ほとんどの MPI 手続では、コミュニケータと呼ばれる引数で、通信を行うプロセスの集合を指定します。プログラム開始時の全てのプロセスに対応するコミュニケータは、MPI の定数 MPI_COMM_WORLD です。各プロセスは、ランクと呼ばれるコミュニケータ中で一意な整数値をもちます。ランクの値は、0 以上、プロセス数-1 以下です。手続 mpi_comm_size および mpi_comm_rank により、それぞれプロセスの個数および自プロセスのランクを取得でき、これらを利用して各プロセスが行う処理をプログラムできます。

MPI プログラムの典型的な例として、図 1.1 の 2 重ループを外側ループで 4 つの仕事に分割し、分散並列化したプログラムを図 3.5 に示します。MPI プログラミングでは、プログラム全体のデータ・処理ではなく、各プロセス用に分割した後のデータ・処理をプログラムします。そのため、定数 ly の値を 25 に変更して（文番号 100）、配列の宣言（文番号 200）とループ長（文番号 300）を分割後の範囲に縮小しています。この際、通信時に受信したデータを配置する領域が必要なので、図 3.6 のように各プロセス上の配列 a の領域を大きめに割り付けていることに注意してください。このような隣接プロセスとの通信のための領域は袖領域と呼ばれます。並列ループ（文番号 300）実行前に、分割境界の処理に必要な配列 a の要素を、一対一通信手続 mpi_sendrecv により隣接プロセス間で送受信（文番号 400）し、並列ループ実行中は、隣接プロセス上のデータの代わりに自プロセス上の袖領域を参照しています。両端のプロセスには片側の隣接プロセスが存在しないので、通信相手を設定する IF 構文（文番号 500）において、通信相手を示す変数 men、mep に、該当するプロセスが存在しないことを示す MPI の定数 MPI_PROC_NULL を設定しています。送受信先として定数 MPI_PROC_NULL を指定すると、その通信は実行されません。

```
      use mpi
      integer :: nproc, me, men, mep
100   integer, parameter :: lx=100, ly=25   ! ly は分割後の寸法
200   real(kind=8) a(0:lx+1,0:ly+1),b(0:lx+1,1:ly), k
   ! 初期化
      call mpi_init(ierr)
   ! プロセスの個数を変数 nproc に取得する
      call mpi_comm_size(MPI_COMM_WORLD,nproc,ierr)
   ! 自プロセスのランクを変数 me に取得する。
      call mpi_comm_rank(MPI_COMM_WORLD,me,ierr)
            :
   ! 通信相手の設定：両端のプロセスは特別処理を行う。
500   if(me .eq. 0)then
        men = MPI_PROC_NULL  ! 下端のプロセスの下隣は存在しない
      else
        men = me - 1  ! 下隣のプロセスのランク
      endif
      if(me .eq. nproc-1)then
        mep = MPI_PROC_NULL  ! 上端のプロセスの上隣は存在しない
      else
        mep = me + 1  ! 上隣のプロセスのランク
      endif
   ! 隣接プロセス間の通信：上方向 → 下方向
400   call mpi_sendrecv(a(1,ly),lx,MPI_DOUBLE_PRECISION,mep,0,a(1,0),       &  ! 上方向
     &     lx,MPI_DOUBLE_PRECISION,men,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE,ierr)
      call mpi_sendrecv(a(1,1),lx,MPI_DOUBLE_PRECISION,men,0,a(1,ly+1),     &  ! 下方向
     &     lx,MPI_DOUBLE_PRECISION,mep,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE,ierr)
   ! 並列化後のループ
300   do j=1,ly
        do i=1,lx
          b(i,j) = a(i,j) + k * (a(i+1,j)-4.0D0*a(i,j)+a(i-1,j)+a(i,j+1)+a(i,j-1))
        enddo
      enddo
            :
   ! 後始末処理
      call mpi_finalize(ierr)
      end
```

図 3.5　MPI プログラム例

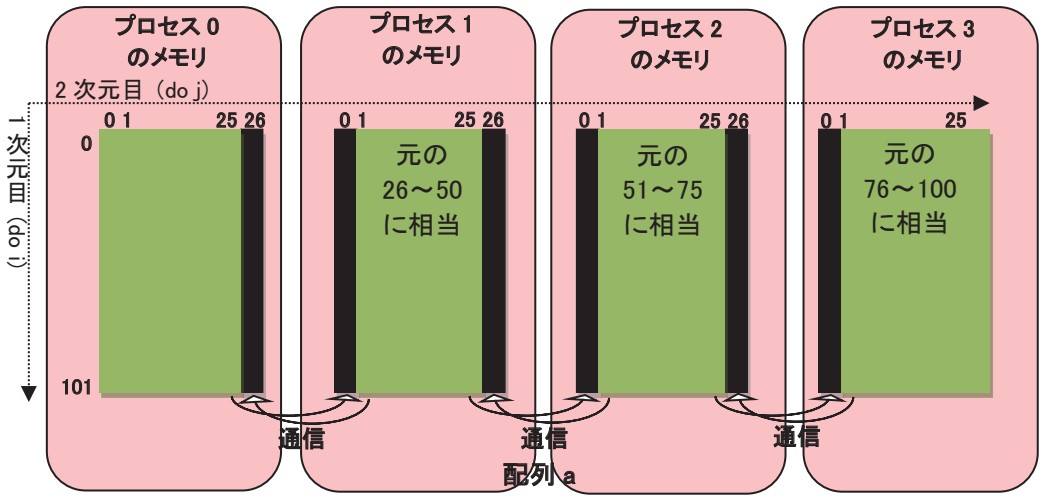


図 3.6　1 次元分割時の配列 a の割付け（黒色は袖領域）

3.3 NEC MPI の活用方法と注意点

3.3.1 MPI プログラムへの標準入力

MPI プログラムに標準入力から入力データを与える場合、図 3.7 のように、実行ファイルに対して直接入力データを与えると、正常に実行できない場合があります。

```
mpirun -np 4 /execdir/a.out < /datadir/inputfile
```

図 3.7　MPI プログラムへの不正な標準入力の例

このような場合、図 3.8 のように、コンパイラの環境変数 VE_FORT5 に入力ファイル名を設定して事前接続した上で、MPI プログラムを実行してください。

```
export VE_FORT5=/datadir/inputfile
mpirun -np 4 /execdir/a.out
```

図 3.8　MPI プログラムへの標準入力

3.3.2 プロセス毎に別々のファイルを対象にした入出力

環境変数 MPIRANK は、コミュニケータとして引数 MPI_COMM_WORLD を指定して、手続 mpi_comm_rank を呼び出すことにより得られるランクと同一の値をもっています。これを利用して、図 3.9 のようなシェルスクリプトを作成し（シェルスクリプトのファイル名を、例えば/execdir/run.sh とします）、図 3.10 のように、mpirun コマンドでそのシェルスクリプトを実行すると、各プロセスのランクと入出力ファイル名とを一対一に対応させ、プロセス毎に別々のファイルを対象にした入出力が可能です。

```
#!/bin/sh
export VE_FORT5=infile.$MPIRANK
export VE_FORT6=outfile.$MPIRANK

exec /execdir/a.out
```

図 3.9　環境変数 MPIRANK を利用した入出力のためのシェルスクリプト例

```
mpirun -np 4 /execdir/run.sh
```

図 3.10　シェルスクリプトを介した MPI プログラムの実行例

プログラム中で接続したファイルに対して入出力する場合は、手続 mpi_comm_rank により取得したランクを利用してファイル名を決めると、やはりプロセス毎に別々のファイルを対象にした入出力が可能です。

3.3.3 プロセス毎に別々のファイルへの標準出力・標準エラー出力

複数の MPI プロセスが同時に標準出力または標準エラー出力を行った場合、それらは入り交じって表示される場合があります。各プロセスの標準出力および標準エラー出力を別々のファイルに容易に出力できるように、NEC MPI はシェルスクリプト/opt/nec/ve/bin/mpisep.sh を用意しています。このスクリプトを使用して、図 3.11 のように MPI プログラムを実行すると、環境変数 NMPI_SEPSELECT の実行時の値に応じて、次のように各プロセスの標準出力および標準エラー出力が別々のファイルに出力されます。なお下記の u は通常 0 となり、r は環境変数 MPIRANK の値となります。

- 環境変数 NMPI_SEPSELECT の値が 1 の場合
  各プロセスの標準出力が、ファイル stdout.u:r に出力されます。

- 環境変数 NMPI_SEPSELECT の値が 2 の場合(既定値)
  各プロセスの標準エラー出力が、ファイル stderr.u:r に出力されます。

- 環境変数 NMPI_SEPSELECT の値が 3 の場合
  各プロセスの標準出力および標準エラー出力が、それぞれファイル stdout.u:r および stderr.u:r に出力されます。

- 環境変数 NMPI_SEPSELECT の値が 4 の場合
  各プロセスの標準出力および標準エラー出力が、ともにファイル std.u:r に出力されます。

```
mpirun -np 4 /opt/nec/ve/bin/mpisep.sh  /execdir/a.out
```

図 3.11　シェルスクリプト /opt/nec/ve/bin/mpisep.sh を使用した実行例

3.3.5 MPI 集団手続デバッグ支援機能

NEC MPI は、実行時でなければ見つけにくい MPI プログラム特有のバグを検出する機能を用意しています。コンパイラオプション-mpiverify を指定して翻訳し、環境変数 NMPI_VERIFY の値を 3 または 4 に設定して実行すると、集団手続に対する呼出し順序の誤りや、引数のプロセス間の不整合などの情報が出力されます。

3.3.6 Fortran からの MPI 使用時の注意点

Fortran プログラム中で引用する非ブロッキングな MPI 手続の実引数として、部分配列、配列式、配列ポインタ、または形状引継ぎ配列を指定すると、引数がメモリ上連続であることを保証するために、コンパイラが一時配列を割り付けて実引数をコピーする場合があります。このとき、実際の引数としては、その一時配列が MPI 手続に渡されます。しかし、その一時配列は、非ブロッキングな MPI 手続の完了以前に解放されてしまうので、プログラムが正常に実行されない可能性があります。非ブロッキングな MPI 手続の実引数には、メモリ上連続な配列を指定してください。形状引継ぎ配列については、コンパイラオプション -fassume-contiguous を指定すると、コンパイラによる一時配列の生成を抑制できます。

3.4 ハイブリッド並列化

複数のプロセスだけによるプログラムの分散並列化をフラット並列化と呼びます。SX-Aurora TSUBASA では、分散並列化と第 2 章でご説明した共有並列化を併用することもできます。これをハイブリッド並列化と呼びます。ハイブリッド並列化は、MPI プログラムを自動並列化または OpenMP により共有並列化するだけで使用可能です。

図 3.5 では、図 1.1 の 2 重ループを外側ループで 4 つの仕事に分割し、4 プロセスに割り当てフラット並列化しました。これを、それぞれ 2 スレッドからなる 2 つのプロセスに割り当てハイブリッド並列化するには、図 3.5 中の定数 ly の値を 50 に変更した上で、自動並列化のためのコンパイラオプション-mparallel を指定して翻訳し、共有並列化します。さらに、環境変数 VE_OMP_NUM_THREADS の値を実行時に 2 に設定すると、各プロセスは 2 スレッドで実行されます。フラット並列化の場合、配列 a を各プロセス上に図 3.6 のように割り付けました。一方、ハイブリッド並列化の場合、配列 a を各プロセス上に図 3.12 のように割り付けることになります。ひとつのプロセス中の複数のスレッドは、そのプロセスの全てのデータ領域を直接参照できるので、スレッド間で通信を行う必要はなく、従ってスレッド間には袖領域も必要ないことに注意してください。

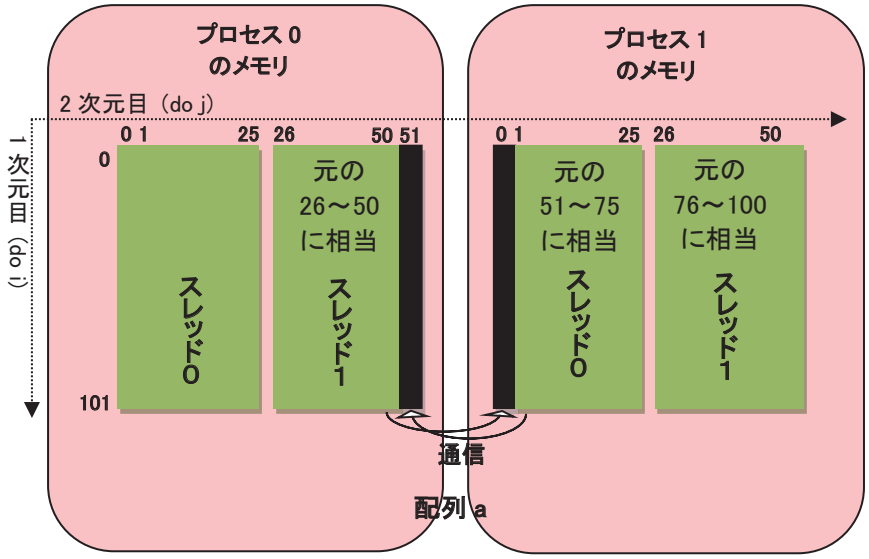


図 3.12　ハイブリッド並列化時の配列 a の割付け（黒色は袖領域）

ひとつの仕事を複数の小さな仕事に分割して、複数のプロセスに割り当てる場合（フラット並列化）と、各プロセス内の複数のスレッドに割り当てる場合（ハイブリッド並列化）とを比較すると、ハイブリッド並列化によりプロセスの総数が削減されます。そのため、プロセス数に依存してコストが増大する通信を含むようなプログラムは、ハイブリッド並列化により経過時間を短縮できる場合があります。その反面、分散並列化と共有並列化の 2 重のオーバヘッドが発生するので、プログラムによってはハイブリッド並列化により経過時間が増加する場合もあります。一方、コンパイラ・MPI の処理系内部のバッファや、全てのプロセスに重複配置するデータの個数は、プロセス数の減少とともに減少するので、ハイブリッド並列化の方がフラット並列化よりも使用メモリの総量を少なくできる傾向があります。

3.4.1　ハイブリッド並列化時の注意点

共有並列化の場合、既定値では各プロセスが VE カード内の全ての CPU コアを使用して共有並列実行を行います。そのため、ハイブリッド並列化時に、ひとつの VE カード上で 2 つ以上のプロセスを実行すると、CPU コアの個数がスレッドの個数より少なくなり、大幅に性能が低下する可能性があります。ハイブリッド並列化時には、各 VE カード上のプロセスの個数を 1 にするか、あるいは共有並列化のスレッドの個数を指定する環境変数 VE_OMP_NUM_THREADS を指定して、各 VE カード上のプロセスの個数とスレッドの個数の積が各 VE カードのコア数（VE30A の場合は 16）以下となるようにしてください。

NEC MPI のスレッドサポートレベルは、MPI_THREAD_SERIALIZED です。従って、全てのスレッドが MPI 手続を呼び出せますが、OpenMP で共有並列化する場合、同一プロセス中の複数のスレッドが同時に MPI 手続を呼び出さないように利用者がプログラムする必要があります。

3.5　性能解析

3.5.1　MPI プログラム実行性能情報

プログラム実行時に環境変数 NMPI_PROGINF の値を、YES、ALL、DETAIL、または ALL_DETAIL に設定することによって MPI プログラムの実行性能情報を採取できます。図 3.13 に、環境変数 NMPI_PROGINF の値を DETAIL に設定して実行した場合の出力例を示します。User Time が最小のプロセス（Min）、最大のプロセス（Max）、および平均（Average）の情報が表示されており、プログラム全体の性能を簡便に把握できます。図中の「LD L3 Hit Element Ratio (%)」は VE30A から追加された情報であり、レベル 3 キャッシュの利用率情報を出力します。

```
MPI Program Information:
========================
Note: It is measured from MPI_Init till MPI_Finalize.
      [U,R] specifies the Universe and the Process Rank in the Universe.
      Times are given in seconds.

Global Data of 4 Vector processes       :          Min [U,R]          Max [U,R]      Average
=================================

Real Time (sec)                         :       64.877 [0,0]       64.878 [0,1]       64.878
User Time (sec)                         :       60.075 [0,0]       64.875 [0,1]       63.675
Vector Time (sec)                       :       54.302 [0,0]       57.173 [0,3]       56.427
Inst. Count                             :  59322128013 [0,0]  64282152733 [0,2]  63012467399
V. Inst. Count                          :  17494773621 [0,0]  17632751395 [0,2]  17593712254
V. Element Count                        : 4423763325401 [0,0] 4424371657756 [0,3] 4424173912218
V. Load Element Count                   : 1302728711714 [0,0] 1302966137322 [0,3] 1302891734718
FLOP Count                              : 1775592960917 [0,1] 1775592961762 [0,0] 1775592961128
MOPS                                    :    77505.531 [0,1]    83610.524 [0,0]    79032.784
MOPS (Real)                             :    77421.537 [0,0]    77504.364 [0,2]    77483.037
MFLOPS                                  :    27369.443 [0,1]    29556.218 [0,0]    27916.186
MFLOPS (Real)                           :    27368.310 [0,1]    27368.418 [0,0]    27368.353
A. V. Length                            :      250.915 [0,2]      252.862 [0,0]      251.466
V. Op. Ratio (%)                        :       99.072 [0,2]       99.167 [0,0]       99.097
L1 Cache Miss (sec)                     :        3.268 [0,1]        3.344 [0,2]        3.292
CPU Port Conf. (sec)                    :        0.000 [0,0]        0.000 [0,0]        0.000
V. Arith. Exec. (sec)                   :       37.520 [0,0]       37.629 [0,1]       37.595
V. Load Exec. (sec)                     :       14.320 [0,0]       16.283 [0,3]       15.708
LD L3 Hit Element Ratio (%)             :       41.257 [0,0]       41.714 [0,3]       41.503
VLD LLC Hit Element Ratio (%)           :       43.127 [0,0]       43.912 [0,3]       43.419
FMA Element Count                       : 557320568364 [0,0] 557320568364 [0,0] 557320568364
Power Throttling (sec)                  :        0.000 [0,0]        0.000 [0,0]        0.000
Thermal Throttling (sec)                :        0.000 [0,0]        0.000 [0,0]        0.000
Memory Size Used (MB)                   :      607.666 [0,0]      609.000 [0,1]      608.000
Non Swappable Memory Size Used (MB)     :       95.667 [0,0]      161.000 [0,1]      112.000


Overall Data of 4 Vector processes
==================================

Real Time (sec)                         :       64.878
User Time (sec)                         :      254.700
Vector Time (sec)                       :      225.707
GOPS                                    :       78.947
GOPS (Real)                             :      309.932
GFLOPS                                  :       27.885
GFLOPS (Real)                           :      109.473
Memory Size Used (GB)                   :        2.375
Non Swappable Memory Size Used (GB)     :        0.438


VE Card Data of 1 VEs
=====================

Memory Size Used (MB) Min               :     2431.998 [node=vh0,ve=0]
Memory Size Used (MB) Max               :     2431.998 [node=vh0,ve=0]
Memory Size Used (MB) Avg               :     2431.998
Non Swappable Memory Size Used (MB) Min :      448.000 [node=vh0,ve=0]
Non Swappable Memory Size Used (MB) Max :      448.000 [node=vh0,ve=0]
Non Swappable Memory Size Used (MB) Avg :      448.000
```

図 3.13　MPI プログラムの実行時性能情報

3.5.2 性能解析機能（FTRACE 機能・NEC Ftrace Viewer）

コンパイラオプション-ftrace を指定して翻訳した MPI プログラムを実行すると、解析情報ファイル ftrace.out.u.r（u は通常 0、r は環境変数 MPIRANK の値）が各プロセスから出力されます。MPI プログラムの場合、コンパイラによる性能解析情報に加えて、通信時間などの情報が得られます。

SX-Aurora TSUBASA が用意する NEC Ftrace Viewer は、性能解析情報をグラフィカルに表示でき、並列プログラムの性能を分析する際にも有効です。例として、8 プロセスにより並列実行したある MPI プログラムの負荷バランスを解析してみます。まず、解析情報ファイルを読み込んだときに表示される Process Breakdown Chart を見ると、図 3.14 のように手続 ITERATE の実行時間（EXCLUSIVE TIME）が大きいことが分かります。手続 ITERATE の負荷バランスを把握するため、左上の「Chart」メニューから Function Statistics Chart を選択します。すると、図 3.15 中の折れ線グラフが示すように、手続 ITERATE の EXCLUSIVE TIME の標準偏差が大きく、負荷バランスが悪いことが分かります。状況をより詳細に調査するため、「Chart」メニューから Process Metrics Chart を選択します。次に、「Metrics Selection」メニューから、MPI COMM. TIME（MPI 通信時間）および MPI IDLE TIME（MPI 待ち時間）を選択します。すると、図 3.16 のように、MPI 通信時間および MPI 待ち時間を示す 2 つの折れ線グラフがほぼ重なって見えます。これは、MPI 通信時間のほとんどが MPI 待ち時間であることを意味しています。また、棒グラフで示される手続 ITERATE の EXCLUSIVE TIME と、折れ線グラフの MPI 待ち時間とを比較すると、EXCLUSIVE TIME に占める MPI 待ち時間が、両端のプロセスだけ短いことが分かります。従って、両端のプロセスの MPI 待ち時間以外の実行時間、つまり演算時間が他のプロセスよりも大きく、相対的に演算時間の小さな他のプロセスにおいて、MPI による通信時に待ちが発生していると推定できます。以上のことから、両端のプロセスの演算を他のプロセスに分配すれば、負荷バランスを改善できる可能性があることが分かります。このように、NEC Ftrace Viewer のグラフ機能を使用すると、負荷バランスなど複数の仕事間の関係を容易に把握できます。

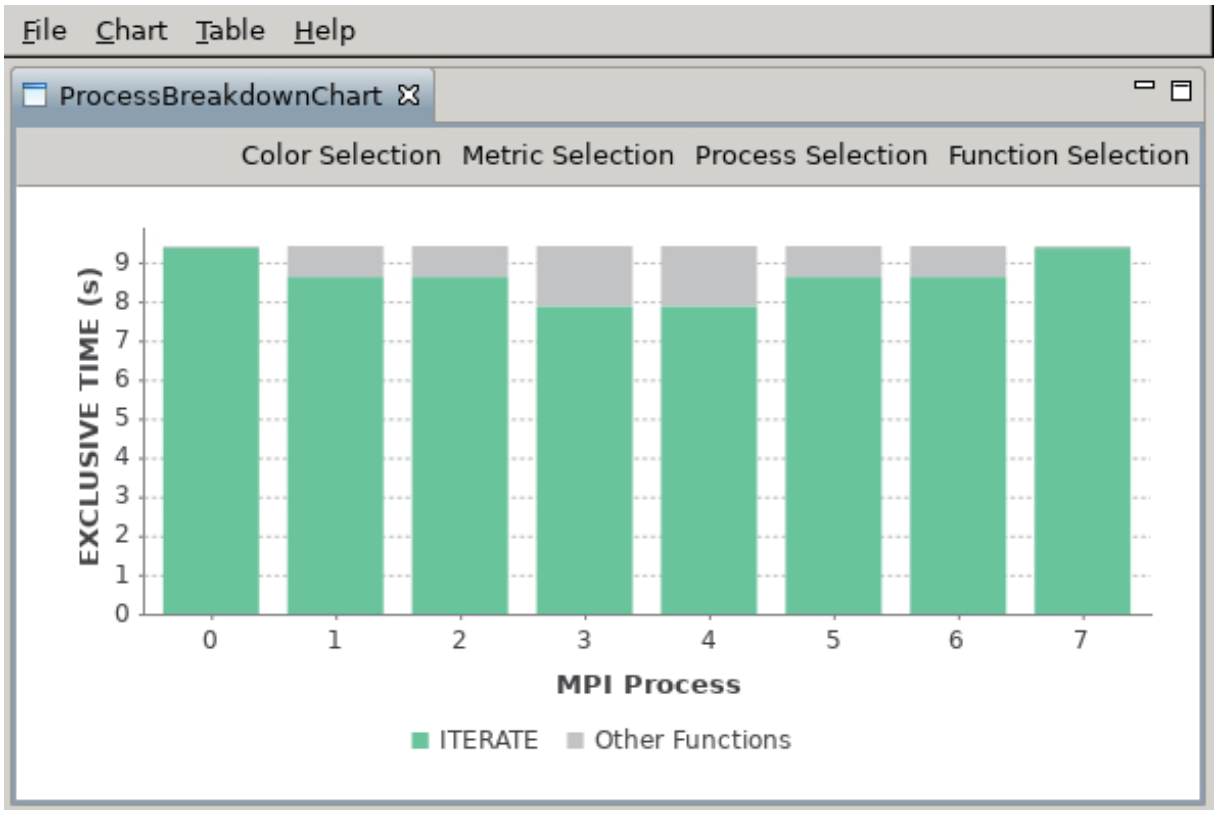


図 3.14　Process Breakdown Chart

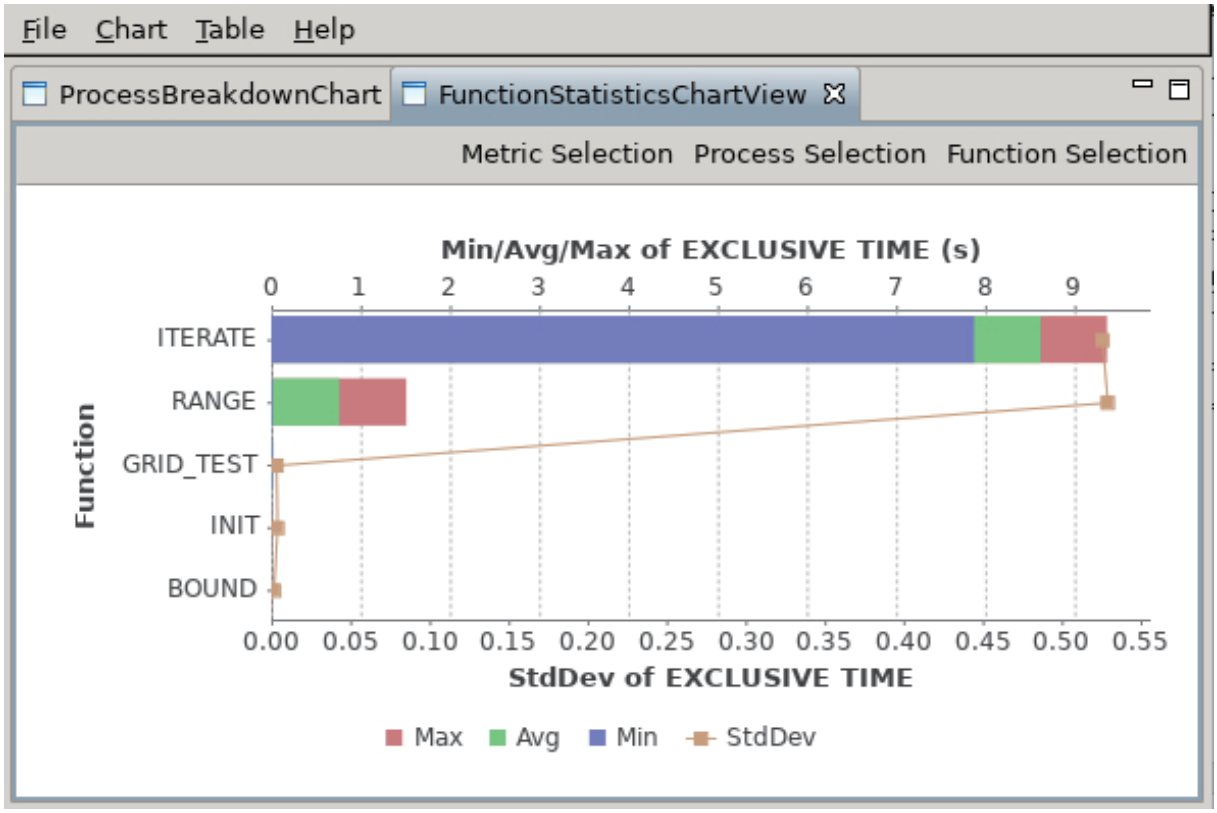


図 3.15　Function Statistics Chart

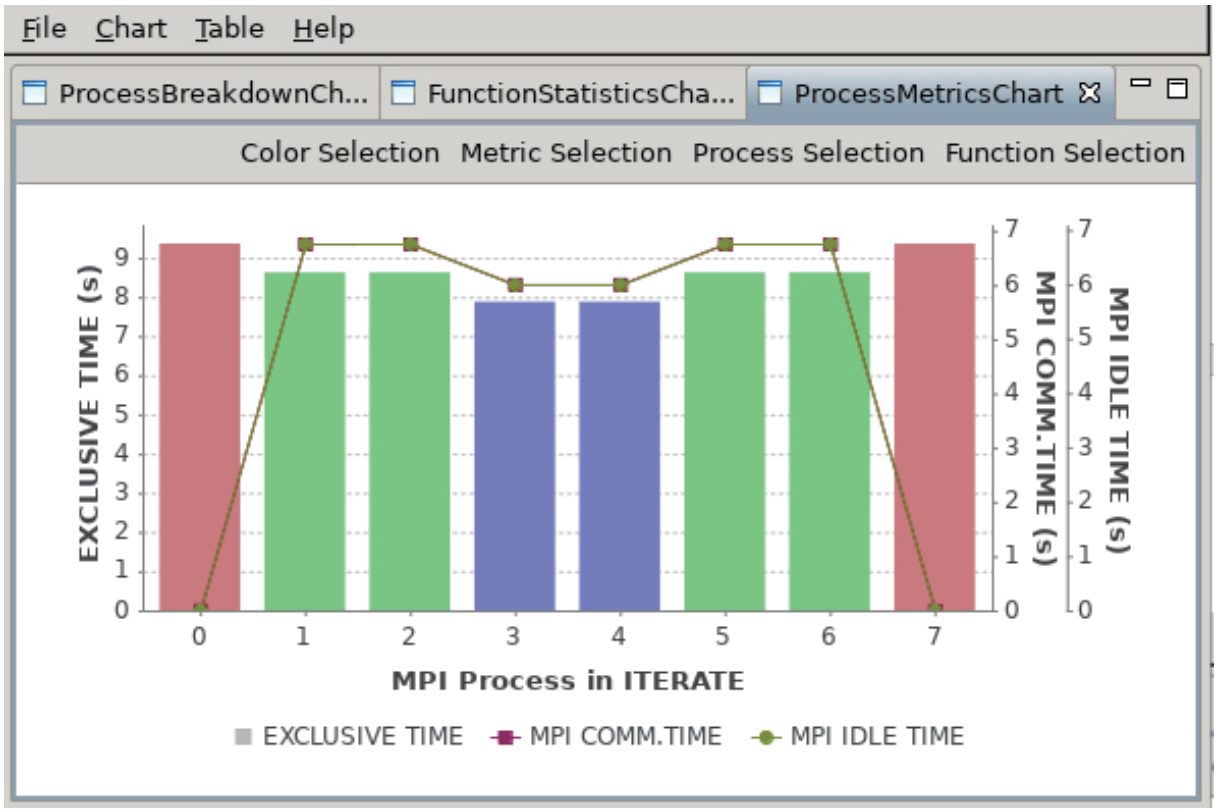


図 3.16　Process Metrics Chart

4. あとがき

本稿では、コンパイラの共有並列化機能 および NEC MPI を使用した分散並列化の基本的事項を中心にご紹介しました。SX-Aurora TSUBASA が用意する並列処理環境の詳細な使用方法については、「Fortran コンパイラ ユーザーズガイド」[3]、「C/C++コンパイラ ユーザーズガイド」[4]、および「NEC MPI ユーザーズガイド」[5] をご覧ください。また、プログラム実行性能情報(PROGINF)および性能解析機能(FTRACE)の詳細な使用方法については、「PROGINF/FTRACE ユーザーズガイド」[6]および「NEC Ftrace Viewer ユーザーズガイド」[7]をご覧ください。SX-Aurora TSUBASA をご使用になる上で、本稿がお役に立てば幸いです。

参考文献

[1] The OpenMP Architecture Review Board http://openmp.org/
[2] Message Passing Interface Forum http://www.mpi-forum.org/
[3] Fortran コンパイラ ユーザーズガイド、日本電気株式会社
https://sxauroratsubasa.sakura.ne.jp/Documentation(Japanese)
[4] C/C++コンパイラ ユーザーズガイド、日本電気株式会社
https://sxauroratsubasa.sakura.ne.jp/Documentation(Japanese)
[5] NEC MPI ユーザーズガイド、日本電気株式会社
https://sxauroratsubasa.sakura.ne.jp/Documentation(Japanese)
[6] PROGINF/FTRACE ユーザーズガイド、日本電気株式会社
https://sxauroratsubasa.sakura.ne.jp/Documentation(Japanese)
[7] NEC Ftrace Viewer ユーザーズガイド、日本電気株式会社
https://sxauroratsubasa.sakura.ne.jp/Documentation(Japanese)

［大規模科学計算システム］

サブシステム AOBA-S の利用法

情報部デジタルサービス支援課 共同利用支援係 共同研究支援係

1　はじめに

本センターはスーパーコンピュータ AOBA のサブシステム AOBA-S の運用を 2023 年 8 月から開始しています。本稿では、サブシステム AOBA-S のプログラミング利用ガイドとして、AOBA-S 用フロントエンドサーバへのログイン、プログラムの作成からコンパイル、およびジョブ実行等の使い方についてご紹介します。

サイバーサイエンスセンターの大規模科学計算システムを利用するためには利用申請が必要です。利用申請について詳しくは以下をご参照ください。

【利用申請】https://www.ss.cc.tohoku.ac.jp/apply-for-use/

2　システムの構成

本センターでは、スーパーコンピュータ AOBA として、サブシステム AOBA-S（SX-Aurora TSUBASA Type 30A）と、サブシステム AOBA-A（SX-Aurora TSUBASA Type 20B）、およびサブシステム AOBA-B（LX 406Rz-2）の 3 つの計算機システムをサービスしています（図 1）。また、AOBA-S 用のストレージとして 4.5PB、AOBA-A および AOBA-B 用のストレージとして 2PB のストレージシステムをサービスしています。利用者は SINET6 を利用したリモートアクセス接続により、全国から大規模科学計算システムを利用することが可能です。

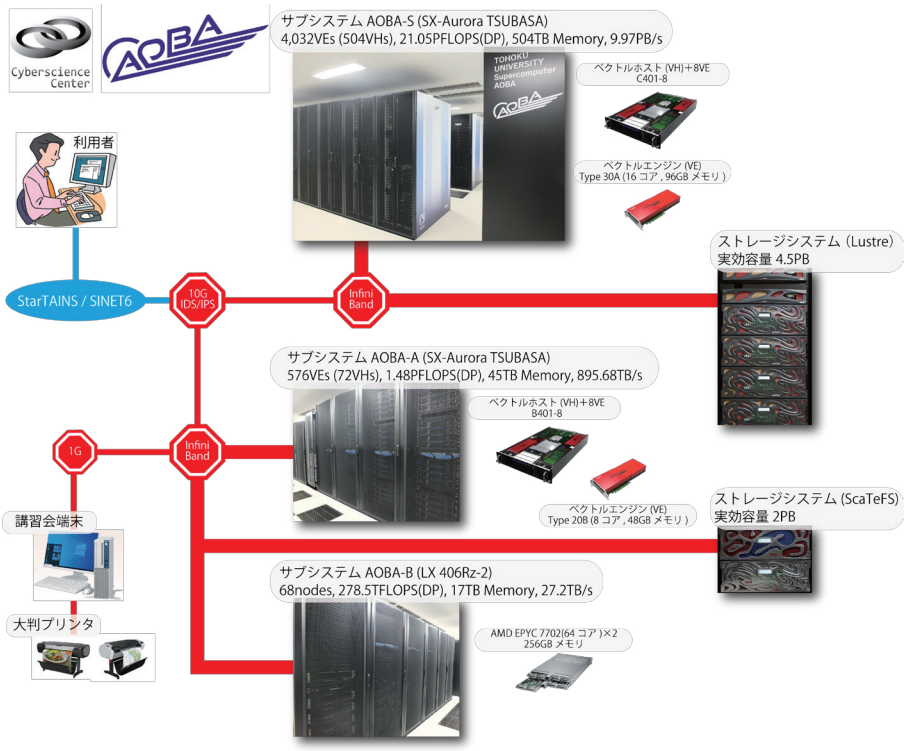


図 1　スーパーコンピュータ AOBA の構成

2.1 サブシステム AOBA-S の特徴

2.1.1 SX-Aurora TSUBASA アーキテクチャ

サブシステム AOBA-S はサブシステム AOBA-A と同じくベクトルアーキテクチャを継承しています。アプリケーション演算処理を行うベクトルエンジン（以下、VE）部と、主に OS 処理を行うベクトルホスト（以下、VH）部により構成されます。PCIe カードに搭載される VE 部はベクトルプロセッサ、および高速メモリから構成され、x86/Linux である VH と PCIe 経由で接続されます。

1VE は理論最大演算性能 4.91TFLOPS(DP) となる 16 コアベクトルプロセッサを 1 基、主記憶は 96GB を搭載し、2.45TB/s という高いメモリバンド幅でプロセッサと接続されることで、高い演算性能とメモリ性能の最適化を実現しています。

今回導入した AOBA-S システムは、1VH と 8VE が構成単位となる C401-8 モデルを採用し、システム全体では 504 個の VH と 4,032 個の VE で構成されます。VE と VH を合わせたシステム全体の理論演算性能は、21.05PFLOPS(DP) 、主記憶は 504TB、総メモリバンド幅は 9.97PB/s となります。

図 2　サブシステム AOBA-S（32 ラック）

- マルチコアベクトルエンジン（16 コア）あたり、4.91TFLOPS(DP) の理論演算性能
- 1VE あたり 96GB の共有メモリを搭載し、メモリバンド幅（データ転送性能）は 2.45TB/s

2.1.2 ベクトルプロセッサ

ベクトルプロセッサとは、ベクトル演算を行う専用のハードウェアを持つコンピュータです。ベクトル演算は、ループ中で繰り返し処理されるような配列データの演算に対して一括して演算を実行するため、高速に演算することができます。

ベクトル計算機は科学技術用の数値計算に適しており、大量のデータを繰り返し処理するような大規模計算に向いていると言われています。本センターでは、流体解析や気象解析、電磁界解析をはじめとする大規模シミュレーションに利用されます。

2.1.3　ソフトウェア

Linux OS 環境上で、ベクトルエンジンの開発環境を利用できます。ベクトル処理、並列処理に対応した大規模プログラムの作成と実行が可能です。

■プログラミング言語　GNU 互換環境を装備し、アプリケーションの実効性能を向上させる高度な自動ベクトル化・自動並列化機能を備えた Fortran/C/C++ コンパイラが利用できます。それぞれのコンパイラは自動ベクトル化機能、自動並列化機能を有していますので、既存のプログラムを修正することなくベクトル化、並列化することができます。自動並列化機能と OpenMP による共有メモリ並列実行と、システム構成に最適化された MPI ライブラリにより、分散メモリ並列実行も可能です。

■科学技術計算ライブラリ　NEC Numeric Library Collection（NLC）は業界標準の BLAS、FFTW、LAPACK、ScaLAPACK を含む、最適化された科学技術計算ライブラリです。NLC は広範な分野の数値シミュレーションプログラムの作成を強力に支援する数学ライブラリのコレクションで、VE での実行に最適化されています。NLC を用いることにより、難解な数値計算アルゴリズムの詳細に煩わされることなく高度な科学技術計算プログラムを作成することができ、数値シミュレーションプログラム開発の生産性を大幅に改善することができます。NLC は、Fortran または C 言語プログラムから利用できます。

2.2　プログラムの実行方法

プログラムの実行の方法として表 1 の 4 種類が利用できます。並列実行には 3 つの手法があります。

■逐次実行　単一のコアで動作するプログラムです。VE 内には 16 コアがありますが、逐次実行では 1 コアのみが利用されます。メモリは 96GB まで利用できます。

■自動並列実行　逐次処理プログラムを、コンパイラが並列化可能な箇所を自動的に判断して並列化します。プログラムを改めて書き直す必要はなく、既存のプログラムをそのまま利用することができます。VE 内の 16 コアまでの並列実行が可能で、メモリは 96GB まで利用できます。

■OpenMP 並列実行　自動並列化と同じく逐次処理プログラムを並列化します。並列化の判断は自動ではなくユーザが明示的に行います。ソース中の並列化したい箇所に並列化指示行を追加します。VE 内の 16 コアまでの並列実行が可能で、メモリは 96GB まで利用できます。

■MPI 並列実行　MPI ライブラリによりプロセッサ間通信を行います。データの分割、処理方法等の並列処理手順を明示的に記述した MPI プログラムを作成する必要があります。分散メモリ並列実行が可能なので、複数の VE を用いた実行が可能です。

センターでは通常のサービスとして最大 2,048VE（32,768 コア）、メモリは 192TB まで利用できます。MPI 並列と自動並列/OpenMP 並列を同時に利用した並列実行も可能です。

表 1　実行の種類、特徴、最大並列数、最大メモリ量

実行の種類	並列化の方法	コードの改変量	最大並列数	最大メモリ量
逐次実行	-	-	1 コア	96GB
自動並列化	コンパイラによる自動並列化	なし	16 コア	96GB
OpenMP 並列	ユーザによる指示行挿入	少ない	16 コア	96GB
MPI 並列	MPI ライブラリを用いたプログラミング	多い	32,768 コア	192TB

3　利用者向けサーバへのログイン方法

AOBA-S 用のフロントエンドサーバ、およびデータ転送サーバへのログインは、AOBA-A,B 用のログインサーバと同じく、公開鍵認証方式による SSH 接続を採用しています。

公開鍵認証方式で使用する鍵ペアの作成方法と、各サーバへのログイン方法についてご紹介します。解説では SSH 接続に以下のターミナル（端末）ソフトを使用する例をご紹介します。

- （Windows の場合） Windows PowerShell
- （macOS ／ Linux の場合） ターミナル

本センターのシステムをはじめて利用する方は以下の手続きが必要です。

(1) 利用者番号の取得（利用申請：https://www.ss.cc.tohoku.ac.jp/apply-for-use/ ）
(2) 鍵ペアの作成（3.3 節）

AOBA-A および AOBA-B を利用していた方は、(1) (2) の手続きは不要です。以前使用していた利用者番号および鍵ペアをそのままご利用いただけます。3.4 節からお読みください。

3.1　利用者向けサーバ名と用途

表 2 に AOBA-S 用の利用者向けサーバ名とホスト名、および用途を示します。

表 2　AOBA-S 用の利用者向けサーバ

サーバ名	ホスト名	用途
フロントエンドサーバ	sfront.cc.tohoku.ac.jp	コンパイル作業、AOBA-S へのジョブ投入 ローカル PC との小規模なデータ転送
データ転送サーバ	sfile.cc.tohoku.ac.jp	ローカル PC との大規模なデータ転送 AOBA-A,B 用ストレージとのデータ転送
HPCI 用フロントエンドサーバ	shpcif.cc.tohoku.ac.jp	HPCI、HPCI-JHPCN 課題用フロントエンドサーバ

3.2　SSH 認証鍵ペアの作成とログインまでの概要

図 3 に、鍵ペアの作成からログインまでの流れを示します。

1. 鍵ペアの作成は 3.3 節で
2. ターミナルソフトの設定および 3. ログイン については 3.4 節で

それぞれ解説します。

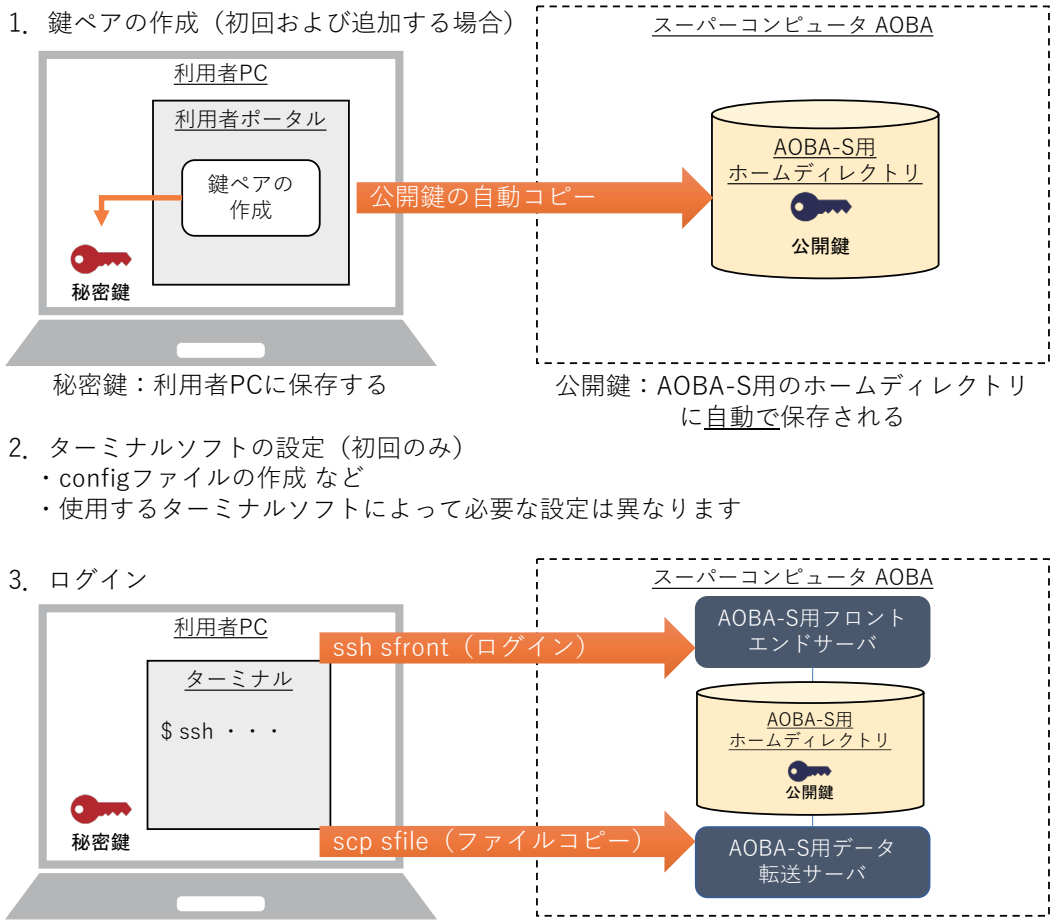


図 3　鍵ペアの作成からログインまで

1. 鍵ペアの作成（初回ログイン時、およびログイン端末を追加する場合）

利用者ポータルで鍵ペアを作成します。作成された秘密鍵は利用者のローカル PC に保存します。公開鍵は、スーパーコンピュータ AOBA のホームディレクトリ上に自動で保存されます。

【利用者ポータル】https://www.ss.cc.tohoku.ac.jp/portal/

2. ターミナルソフトの設定（初回ログイン時）

各利用者用サーバにログインするための設定を行います。使用するターミナルソフトによって必要な設定が異なります。

3. ログイン

利用者のローカル PC に保存した秘密鍵を使って利用者用サーバにログインします。

3.3 公開鍵認証方式で使用する鍵ペアの作成

3.3.1 公開鍵認証方式を使用する上での注意事項

以下の注意事項を必ず守ってください。

守られない場合、不正アクセス（不正ログイン、クライアントのなりすまし、暗号化された通信の暴露、他サーバへの攻撃など）のリスクが非常に高まるので**大変危険**です。

- パスフレーズなしの秘密鍵を使用しないこと
- 秘密鍵、パスフレーズを使いまわさないこと
- 秘密鍵を持ち出さないこと（メールに添付しない、USB メモリ等に保存しない）
- 秘密鍵をスーパーコンピュータ AOBA のホームディレクトリに保存しないこと
- 公開鍵と秘密鍵の鍵ペアを同一ホスト上に保存しないこと

3.3.2 鍵ペアの作成（初回ログイン時、および接続する PC を追加する場合）

■初回ログイン時　鍵ペアの作成は利用者ポータルで行います。

(1) 以下の URL から利用者ポータルに接続します。利用者ポータルには利用者番号と LDAP パスワード（※）でログインします。
【利用者ポータル】https://www.ss.cc.tohoku.ac.jp/portal/
(2)「SSH 公開鍵登録」ボタンをクリックします。
(3) 利用者ポータルの画面の説明に従い、鍵ペアを作成します。
（パスフレーズを設定し、鍵生成・登録ボタンをクリック）
(4) 作成された秘密鍵を利用者のローカル PC に保存します。保存先は以下を推奨します。フォルダがない場合は新規作成します。
  - （Windows の場合） C:¥Users¥（ユーザ名）¥.ssh
  - （macOS ／ Linux の場合） $HOME/.ssh

ポータルサイトで作成された公開鍵は、AOBA-S と AOBA-A,B のホームディレクトリの公開鍵ファイル（$HOME/.ssh/authorized_keys）に自動で保存されます。

※利用者ポータルで使用する LDAP パスワードの変更方法は、3.6 節を参照してください。

■別の PC からログインする場合　接続する PC からポータルサイトにアクセスし、新しい鍵ペアを作成します。既存の秘密鍵を使いまわすのではなく、端末ごとに鍵ペアを作成してください。

3.4 利用者向けサーバへのログイン方法

3.4.1 ターミナルソフトの設定（各 PC での設定）

利用者の PC 上で、ターミナルソフトの設定を行います。以降の解説では、次のフォルダを「.ssh フォルダ」と呼び、秘密鍵を「id_rsa_cc」というファイル名で.ssh フォルダに保存した場合とします。

- （Windows の場合） C:¥Users¥（ユーザ名）¥.ssh
- （macOS ／ Linux の場合） $HOME/.ssh

各利用者向けサーバのホスト名は、次の文字列で設定するものとして解説します。ホスト名には任意の文字列を設定することができますが、他の文字列を設定した場合は、以降の解説におけるホスト名を読み替えてください。

- （AOBA-S 用フロントエンドサーバ） sfront
- （AOBA-S 用データ転送サーバ） sfile
- （AOBA-S HPCI 用フロントエンドサーバ） shpcif

(1) macOS ／ Linux の場合は、秘密鍵のパーミッションを 600 に変更が必要です。ターミナルソフトを起動し以下のコマンドを実行します（リスト 1)。

リスト 1　秘密鍵のパーミッション変更

```
(localhost)$ chmod 600 ${HOME}/.ssh/id_rsa_cc
```

以降は Windows、macOS ／ Linux 共通です。

(2) .ssh フォルダの「config」というファイルをテキストエディタで開きます。ファイルがない場合は新規作成します。拡張子は付けません。（フォルダの設定を「拡張子を表示しない」にしている場合、意識せずに拡張子つきのファイルを作成している可能性があります。config ファイルに拡張子がついていると正しく設定が読み込まれません。）

(3) config ファイルにリスト 2 に示した設定を記述します。ホスト名を例とは別に設定した場合は、以降のコマンドではご自身の環境に合わせて読み替えてください。

リスト 2　config ファイルの設定方法

```
# AOBA-S用フロントエンドサーバに接続するための設定
Host sfront                          # AOBA-S用フロントエンドサーバ名をsfrontで指定
HostName sfront.cc.tohoku.ac.jp      # ホスト名をFQDNで指定
User 利用者番号                      # 利用者番号を指定
IdentityFile $HOME/.ssh/id_rsa_cc    # 秘密鍵の保存場所とファイル名を指定

# AOBA-S用データ転送サーバに接続するための設定
Host sfile                           # AOBA-S用データ転送サーバ名をsfileで指定
HostName sfile.cc.tohoku.ac.jp       # ホスト名をFQDNで指定
User 利用者番号                      # 利用者番号を指定
IdentityFile $HOME/.ssh/id_rsa_cc    # 秘密鍵の保存場所とファイル名を指定

# HPCI用フロントエンドサーバに接続するための設定
Host shpcif                          # HPCI用フロントエンドサーバ名をshpcifで指定
HostName shpcif.cc.tohoku.ac.jp      # ホスト名をFQDNで指定
User 利用者番号                      # 利用者番号を指定
IdentityFile $HOME/.ssh/id_rsa_cc    # 秘密鍵の保存場所とファイル名を指定
```

3.4.2　AOBA-S 用フロントエンドサーバへのログイン

ターミナルソフトを起動しリスト 3 に示したコマンドを実行すると config ファイルに記述した設定が読み込まれ、AOBA-S 用のフロントエンドサーバにログインします。

リスト 3　AOBA-S 用フロントエンドサーバへのログイン

```
(localhost)$ ssh sfront
Last login: Tue Aug 1 12:34:56 2023 from x.x.x.x
(sfront)$ # 接続するとプロンプトのホスト名が変わります
```

フロントエンドサーバは冗長構成になっており、自動的に sfront1 または sfront2 が選択されます。どちらにログインしても環境は変わりません。

なお、フロントエンドサーバでは一定時間以上のプロセスは実行できません。また、大容量のデータ転送はシステムに高い負荷がかかります。大容量のデータ転送を行う場合はデータ転送サーバをご利用ください。

3.4.3 AOBA-S 用データ転送サーバの利用方法

AOBA-S 用のデータ転送サーバは、利用者のローカル PC と AOBA-S 用のストレージ間のデータ転送、および AOBA-S 用ストレージと AOBA-A,B 用ストレージ間のデータコピーに用います。

ローカル PC とのデータ転送は、scp コマンドや sftp コマンド、ファイル転送のアプリケーションを用いて行います。リスト 4 に scp コマンドを使用したデータ転送の例を示します。ターミナルソフトを起動し、scp コマンドでローカル PC と AOBA-S 用ストレージ間のデータ転送を行います。

リスト 4　AOBA-S 用データ転送サーバ

```
# ローカルPCのデータをAOBA-S用ストレージに転送する場合
(localhost)$ scp -r ローカルPCのデータ sfile:/uhome/利用者番号/コピー先

# AOBA-S用ストレージのデータをローカルPCに転送する場合
(localhost)$ scp -r sfile:/uhome/利用者番号/データ ローカルPCのコピー先
```

また AOBA-S 用ストレージと AOBA-A,B 用ストレージ間のデータコピーは、AOBA-S 用のデータ転送サーバにログインした後に cp コマンドで行います。リスト 5 に AOBA-S 用のデータ転送サーバへのログイン方法、およびリスト 6 に cp コマンドを使用したデータコピーの例を示します。

ターミナルソフトを起動しリスト 5 示したコマンドを実行すると config ファイルに記述した設定が読み込まれ、AOBA-S 用のデータ転送サーバにログインします。

リスト 5　AOBA-S 用データ転送サーバ

```
(localhost)$ ssh sfile
Last login: Tue Aug 1 12:34:56 2023 from x.x.x.x
(sfile)$ # 接続するとプロンプトのホスト名が変わります
```

AOBA-S 用データ転送サーバ上でのマウントポイントは以下の通りです。

- （AOBA-S 用ストレージ）　/uhome/利用者番号/
- （AOBA-A,B 用ストレージ）/mnt/stfs/uhome/利用者番号/

sfile にログインした後リスト 6 に示したコマンドで、AOBA-S 用ストレージと AOBA-A,B 用ストレージ間のデータコピーを行います。

リスト 6　AOBA-S 用データ転送サーバ

```
# AOBA-A,B用ストレージのデータをAOBA-S用ストレージにコピーする場合
(sfile)$ cp /mnt/stfs/uhome/利用者番号/AOBA-A,Bのデータ /uhome/利用者番号/AOBA-Sのコピー先

# AOBA-S用ストレージのデータをAOBA-A,Bストレージにコピーする場合
(sfile)$ cp /uhome/利用者番号/AOBA-Sのデータ /mnt/stfs/uhome/利用者番号/AOBA-A,Bのコピー先
```

なお、AOBA-S 用フロントエンドサーバと HPCI 用フロントエンドサーバからは、AOBA-A,B 用ストレージ用のマウントポイントにはアクセスできません。

3.4.4 HPCI 用フロントエンドサーバへのログイン

ターミナルソフトを起動しリスト 7 に示したコマンドを実行すると config ファイルに記述した設定が読み込まれ、HPCI 用のフロントエンドサーバにログインします。HPCI および HPCI-JHPCN 課題利用者のみログイン可能です。

リスト 7　HPCI 用フロントエンドサーバ

```
(localhost)$ ssh shpcif
Last login: Tue Aug 1 12:34:56 2023 from x.x.x.x
(shpcif)$ # 接続するとプロンプトのホスト名が変わります
```

3.5　ログインシェルの確認と変更

ログインシェルのデフォルトは bash が設定されています。設定の確認および変更はリスト 8 に示した手順で行います。ログインシェルの変更がシステム全体に反映されるまで、15 分程度かかります。

ログインシェルの変更は、AOBA システム（AOBA-S 用、AOBA-A,B 用の全てのホスト、プリンタサーバ）の利用者向けサーバに対して設定されます。

リスト 8　ログインシェルの確認と変更

```
(sfront)$ fchsh                    （現在のログインシェルの確認）
Enter Password:                    （LDAPパスワードを入力）
loginShell: /bin/bash              （現在のログインシェルが表示される）

(sfront)$ fchsh /bin/tcsh          （ログインシェルを/bin/tcshに変更）
Enter Password:                    （LDAPパスワードを入力）
Changed loginShell to /bin/tcsh    （ログインシェルが変更された）
```

3.6　LDAP パスワードの変更

利用者ポータルやログインシェルの変更などで使用する LDAP パスワードの変更は、利用者ポータルで行います。

(1) 以下の URL から利用者ポータルにログインします。利用者ポータルには、利用者番号と現在の LDAP パスワードでログインします。
【利用者ポータル】https://www.ss.cc.tohoku.ac.jp/portal/
(2)「パスワード変更」ボタンをクリックします。
(3) 利用者ポータルの画面の説明に従い、新しい LDAP パスワードを設定します。
(4) 以下で使用するパスワードが変更されます。
- 利用者ポータルへのログイン
- 大判カラープリンタのプリンタサーバへのログイン
- ログインシェルの変更時

4　プログラムのコンパイルと実行

本章ではプログラムのコンパイルと、サブシステム AOBA-S で実行する手順を説明します。

4.1　AOBA-S 用フロントエンドサーバへのログイン

サブシステム AOBA-S 用プログラムのコンパイルとジョブ投入は AOBA-S 用フロントエンドサーバで行います。AOBA-S 用フロントエンドサーバへの詳しい接続方法は 3.4 節をご参照ください。設定が完了している場合はリスト 9 に示したコマンドで、AOBA-S 用フロントエンドサーバにログインします。

なお AOBA-S 用フロントエンドサーバから、AOBA-A および AOBA-B へのジョブ投入などの操作はできません。

リスト 9　AOBA-S 用フロントエンドサーバへのログイン

```
(localhost)$ ssh sfront
Last login: Tue Aug 1 12:34:56 2023 from x.x.x.x
(sfront)$ # 接続するとプロンプトのホスト名が変わります
```

4.2　ソースコードの作成

ソースコードを作成する代表的な方法は以下の通りです。

- フロントエンドサーバ上でテキストエディタで作成する
- ローカル PC 上のテキストエディタで作成し、AOBA-S 用ストレージにファイル転送する
- Visual Studio Code などの開発環境で作成する

フロントエンドサーバのコンソール上でソースファイルを作成するためのテキストエディタは、emacs や vim、nano が利用できます。

ローカル PC で作成したソースコードファイルを転送する場合は、ファイル転送ソフトや scp コマンドで利用者のホームディレクトリに転送してください。その際、ソース（テキスト）ファイルは ASCII モードで転送します。

Windows 環境で作成したソースコードの改行コードと文字コードを Linux 用に変換するためには、AOBA-S 用ストレージに転送したソースコードに nkf コマンドを利用します（リスト 10）。

リスト 10　nkf コマンドの使用例

```
# 改行コードをLF、文字コードをUTF-8に変換し、別のファイルに書き出す方法
(sfront)$ nkf -Lu ソースコードファイル名 > 変換後ファイル名

# 改行コードをLF、文字コードをUTF-8に変換して上書き保存する方法
(sfront)$ nkf --overwrite -Lu ソースコードファイル名
```

開発環境の利用方法については各アプリケーションのマニュアルをご参照ください。

4.3　コンパイル

4.3.1　コンパイラの仕様

AOBA-S では SX-Aurora TSUBASA 用に最適化された Fortran コンパイラ、および C/C++ コンパイラを利用できます。コンパイラの仕様は以下の通りです。

Fortran コンパイラ

- nfort、mpinfort コマンドでコンパイルとリンク
- 自動ベクトル化・自動並列化機能を実装
- ISO/IEC 1539-1:2010 Programming languages - Fortran に準拠
- OpenMP Application Program Interface Version 4.5 に準拠
- ISO/IEC 1539-1:2018 Programming languages - Fortran の一部機能にも対応
- OpenMP Application Program Interface Version 5.0 の一部機能にも対応

C/C++ コンパイラ

- ncc、mpincc、nc++、mpinc++ コマンドでコンパイルとリンク
- 自動ベクトル化・自動並列化機能を実装
- ISO/IEC 9899:2011 Programming languages - C に準拠
- ISO/IEC 14882:2014 Programming languages - C++ に準拠
- ISO/IEC 14882:2017 Programming languages - C++ に準拠
- OpenMP Application Program Interface Version 4.5 に準拠
- ISO/IEC 14882:2020 Programming languages - C++ の一部機能にも対応
- OpenMP Application Program Interface Version 5.0 の一部機能にも対応

4.3.2　コンパイルの手順

ソースコードファイルはそれぞれの言語用コマンドでコンパイルを行います。以下では単一コアで実行する逐次実行、自動並列化または OpenMP 並列化による共有メモリ並列実行、および MPI ライブラリによる分散メモリ並列実行のコンパイル手順について解説します。

■Fortran プログラムのコンパイル　ソースコードファイルは nfort コマンドまたは mpinfort コマンドでコンパイルを行います。必要に応じて、表 3 に示したコンパイルオプションをコンパイル時に指定します。その他のコンパイルオプションについては、6 章のコンパイラユーザーズガイドをご参照ください。

ソースファイルの拡張子は、自由形式（フリーフォーマット）なら.f90 .f95 か.F90 .F95 を、固定形式（7 カラム目から記述）なら.f か.F を、2003 形式であれば.f03 か.F03 を付けます。（拡張子が大文字で始まるものは、コンパイルの前に fpp によるプリプロセス処理が行われます。）

コンパイルが成功すると出力ファイル名を指定しない場合は、カレントディレクトリに実行モジュールの a.out が作成されます。

表 3　Fotran,C/C++ コンパイラの主なオプション

コンパイルオプション	機能
-On	最適化レベルを指定する
n に指定できる値	
4	言語仕様を逸脱した副作用を伴う最大限の最適化・自動ベクトル化を適用する
3	副作用を伴う最適化・自動ベクトル化、および、多重ループの最適化を適用する
2	（既定値）副作用を伴う最適化・自動ベクトル化を適用する
1	副作用を伴わない最適化・自動ベクトル化を適用する
0	最適化、自動ベクトル化、並列化、インライン展開を適用しない （最適化レベルを高くすると、最適化の副作用により計算結果が変わることが ありますのでご注意ください）
-finline-functions	自動インライン展開機能を適用する
-mparallel	自動並列化機能を利用する
-fopenmp	OpenMP 並列化を利用する
-mno-parallel-omp-routine	OpenMP ディレクティブを含むルーチンを自動並列化しない （-mparallel と-fopenmp が同時に指定されたとき、ループが OpenMP の並列区間の 外側にある場合は外側のループが自動並列化の対象となります）
-ftrace	性能解析 ftrace 機能用の実行ファイルを作成する
-report-diagnostics	ベクトル診断リストを出力する
-fdiag-vector=2	詳細なベクトル化診断メッセージを出力する（既定値は 1）
-report-format	ベクトル化、並列化などの最適化情報がソース行とともに出力された編集リストを 出力する
-report-all	コード生成リスト、診断メッセージリスト、編集リスト、インラインリスト、 オプション リスト、ベクトルリストを出力する
-fcheck=bounds	バウンスチェック（配列の上下限のチェック）を行う
-fcheck=all	仮引数のエリアスへの代入、ビット intrinsic な引数、配列の上下限、 不正なポインタ、DO ループのステップ値が 0 かどうか、整数オーバーフロー、 ポインタ参照、省略可能な引数の参照、不正な再帰呼出しのチェックを行う

リスト 11　nfort コマンドの使用例

```
（逐次実行）
(sfront)$ nfort コンパイルオプション Fortranソースファイル名

（自動並列化実行）
(sfront)$ nfort -mparallel コンパイルオプション Fortranソースファイル名

（OpenMP並列実行）
(sfront)$ nfort -fopenmp コンパイルオプション Fortranソースファイル名
```

リスト 12　mpinfort コマンドの使用例

```
（MPI並列実行）
(sfront)$ mpinfort コンパイルオプション Fortranソースファイル名

（MPIと自動並列化の同時並列実行）
(sfront)$ mpinfort -mparallel コンパイルオプション Fortranソースファイル名

（MPIとOpenMPの同時並列実行）
(sfront)$ mpinfort -fopenmp コンパイルオプション Fortranソースファイル名
```

■C/C++ プログラム　ソースコードは ncc または mpincc コマンドで C プログラムを、nc++ または mpinc++ コマンドで C++ プログラムをコンパイルします。必要に応じて Fortran コンパイラと同じく、表 3 に示したコンパイルオプションをコンパイル時に指定します。その他のコンパイルオプションについては、6 章のコンパイラユーザーズガイドをご参照ください。

ソースファイルの拡張子は、C 言語であれば.c を、C++ であれば.C .cc .cpp .cp .cxx .c++ のいずれかを付けます。

コンパイルが成功すると出力ファイル名を指定しない場合は、カレントディレクトリに実行モジュールの a.out が作成されます。

リスト 13　ncc/nc++ コマンドの使用例

```
（逐次実行）
(sfront)$ ncc コンパイルオプション Cソースファイル名
(sfront)$ nc++ コンパイルオプション C++ソースファイル名

（自動並列化実行）
(sfront)$ ncc -mparallel コンパイルオプション Cソースファイル名
(sfront)$ nc++ -mparallel コンパイルオプション C++ソースファイル名

（OpenMP並列実行）
(sfront)$ ncc -fopenmp コンパイルオプション Cソースファイル名
(sfront)$ nc++ -fopenmp コンパイルオプション C++ソースファイル名
```

リスト 14　mpincc/mpinc++ コマンドの使用例

```
（MPI並列実行）
(sfront)$ mpincc コンパイルオプション Cソースファイル名
(sfront)$ mpinc++ コンパイルオプション C++ソースファイル名

（MPIと自動並列化の同時並列実行）
(sfront)$ mpincc -mparallel コンパイルオプション Cソースファイル名
(sfront)$ mpinc++ -mparallel コンパイルオプション C++ソースファイル名

（MPIとOpenMPの同時並列実行）
(sfront)$ mpincc -fopenmp コンパイルオプション Cソースファイル名
(sfront)$ mpinc++ -fopenmp コンパイルオプション C++ソースファイル名
```

■実行時性能解析情報（FTRACE）の出力　コンパイル時に-ftrace オプションを指定すると、実行後にディレクトリに性能解析情報の結果ファイル（ftrace.out）が書き出されます。MPI 並列実行時の性能情報も取得可能です。

性能解析情報の確認は、フロントエンドサーバ上で ftrace コマンドを実行してテキスト形式で確認するか、ftraceviewer コマンドで GUI で確認することができます。図 4 は Ftrace Viewer の表示例です。

FTRACE と Ftrace Viewer についての詳細は、6 章の PROGINF/FTRACE ユーザーズガイド、および NEC Ftrace Viewer ユーザーズガイドをご参照ください。

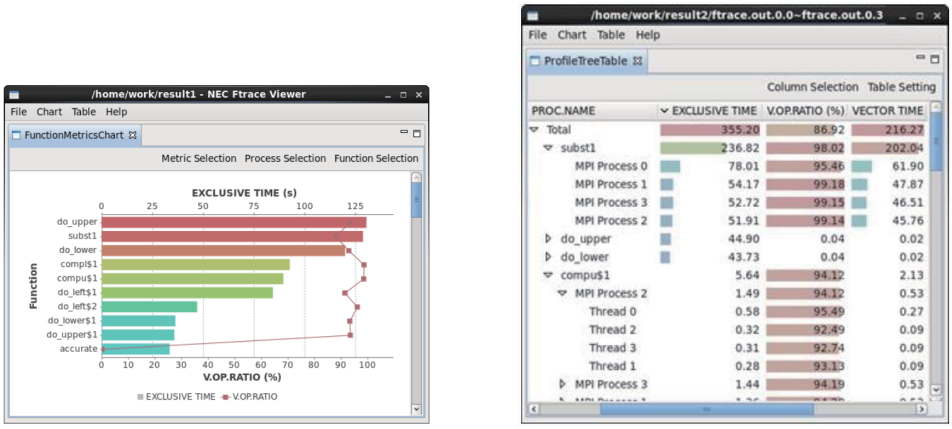


図 4　Ftrace Viewer の表示例

4.4　バッチリクエストによるジョブの実行

フロントエンドサーバでコンパイル作業を行って作成したプログラムの実行は、バッチ処理と呼ばれる方法で計算機に実行を依頼します。本センターではバッチ処理に NEC Network Queuing System V（以下、NQSV）を採用しています。

NQSV についてのコマンドやオプションについての詳細は、「NQSV 利用の手引 操作編」をご参照ください。なお、当センター独自の運用方法のためマニュアルの記載の通りに動作しない場合もあります。

4.4.1　バッチリクエストの概要

バッチリクエストは、バッチ処理で計算機にジョブの実行を依頼するリクエストのことです。ジョブとは、実行可能ファイル、プログラム、実行モジュールまたはコマンドのことで、リクエストは 1 つまたは複数のジョブから構成されます。

図 5 にバッチリクエスト実行の概念図を示します。

1. ジョブスクリプトの作成 は 4.4.3 から 4.4.5 節で
2. バッチリクエストの投入 は 4.4.6 節で
3. 実行待ち→実行 は 4.4.7 から 4.4.9 節で
4. 実行終了 は 4.4.10 でそれぞれ解説します。

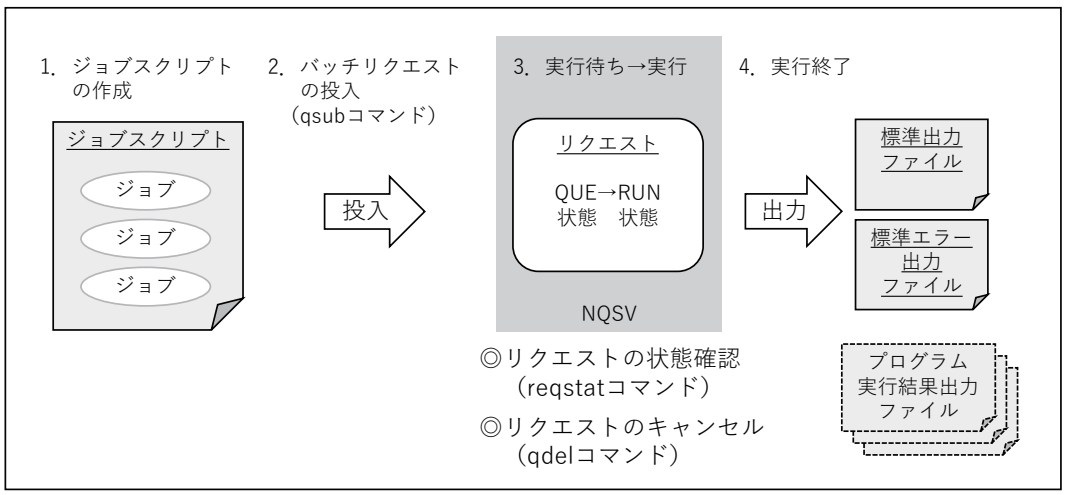


図 5　バッチリクエストの概念図

1. ジョブスクリプトの作成

プログラムの実行手続きを書いたテキストファイルを作成します。利用形態、利用 VE 数、最大経過時間も記述します。ノードの空き状況によっては最大経過時間をデフォルトの時間よりも短く指定することで、バックフィルスケージューリングにより実行開始予定時刻が早くなることがあります。

2. バッチリクエストの投入

3. 実行待ち→実行

以下のコマンドでバッチリクエストを操作します。

- qsub コマンド　　NQSV にバッチリクエストの投入
- reqstat コマンド　投入したリクエストの状態を表示（状態確認）
- qdel コマンド　　投入したリクエストのキャンセル

4. 実行終了

リクエストの実行が終了すると、標準出力と標準エラー出力がファイルとして書き出されます。標準出力/標準エラー出力のファイルサイズ上限は 1GB です。ファイルサイズが上限値を超えた場合はプログラムが強制終了しますので、実行結果はファイル名を指定して書き出しを行ってください。

終了したリクエストは reqstat コマンドの表示から削除されます。

4.4.2　キュー構成

リクエストの投入キューについて説明します。

表 4 に AOBA-S のキュー構成を示します。ジョブスクリプトの冒頭で、表に示したキュー名、利用 VE 数、最大経過時間を指定します。

表 4　サブシステム AOBA-S のキュー構成

利用形態	キュー名	VE 数	最大経過時間 規定値/最大値	メモリサイズ
無料	sxsf	1	1 時間/1 時間	96GB
共有	sxs	1～2,048	72 時間/720 時間	96GB × VE 数
占有	個別設定			96GB × VE 数

経過時間は、バッチリクエストが開始してから終了するまでの時間です。指定した最大経過時間を超えた場合、プログラムの実行中でもバッチリクエストは強制的に終了します。I/O が高負荷となる場合、経過時間が想定よりも遅くなる場合がありますので、必要十分な最大経過時間を指定してください。

なお、最大経過時間は最大値 720 時間（720:00:00）を超えて指定することは出来ません。（sxf では経過時間の最大値は 1:00:00 です。）

4.4.3　バッチリクエストの作成

バッチリクエスト用のシェルスクリプトファイル（ジョブスクリプト）を作成します。通常のシェルスクリプトと同様に、テキストファイル形式で任意のコマンドを組み合わせてプログラムの実行手続きを記述します。ジョブスクリプトを実行するシェルは、sh、csh ともに使用できますが、解説では sh スクリプト形式で記述し、ファイル名を run.sh とします。

ジョブスクリプトに記述する基本項目は以下の通りです。その他に、環境変数の指定やファイル操作コマンドが必要な場合は、適切な箇所に記述します。

- 投入キュー名
- 利用する VE 数
- 最大経過時間
- 作業ディレクトリへの移動コマンド
- プログラムの実行コマンド

4.4.4 ジョブスクリプトの例

リスト 15〜リスト 20 に AOBA-S 向けのジョブのスクリプトファイル例を示します。

■逐次実行の場合 リスト 15 およびリスト 16 に、逐次実行の場合のジョブスクリプトの例を示します。逐次実行の場合、プログラムは 1VE 内の 1 コアで実行されます。

リスト 15 ジョブスクリプトの例（逐次実行）

```
#!/bin/sh
#PBS -q sxs                     # AOBA-Sを使用する
#PBS --venode 1                 # VEを1個使う
#PBS -l elapstim_req=2:00:00    # 最大経過時間を2時間に指定

cd $PBS_O_WORKDIR               # qsubを実行したディレクトリに移動
./a.out                         # カレントディレクトリの a.out を実行
```

リスト 16 ジョブスクリプトの例（逐次実行、無料キュー）

```
#!/bin/sh
#PBS -q sxsf                    # AOBA-Sの無料キューを使用する
#PBS --venode 1                 # VEを1個使う
#PBS -l elapstim_req=0:30:00    # 最大経過時間を30分に指定

cd $PBS_O_WORKDIR               # qsubを実行したディレクトリに移動
./a.out                         # カレントディレクトリの a.out を実行
```

■自動並列化/OpenMP 並列実行の場合 リスト 17 に自動並列化/OpenMP 並列実行の場合のジョブスクリプトの例を示します。自動並列化/OpenMP 並列実行の場合、プログラムは 1VE 中の 2〜16 コアで実行されます。実行コア数の指定は環境変数 VE_OMP_NUM_THREADS で行います。逐次実行と同様に、sxf も指定できます。

リスト 17 ジョブスクリプトの例（自動並列化/OpenMP 並列実行）

```
#!/bin/sh
#PBS -q sxs                     # AOBA-Sを使用する
#PBS --venode 1                 # VEを1個使う
#PBS -l elapstim_req=2:00:00    # 最大経過時間を2時間に指定
#PBS -v VE_OMP_NUM_THREADS=16   # 16コア並列で実行

cd $PBS_O_WORKDIR               # qsubを実行したディレクトリに移動
./a.out                         # カレントディレクトリの a.out を実行
```

■MPI 並列実行の場合 リスト 18 に MPI 並列実行の場合のジョブスクリプトの例を示します。MPI 並列実行の場合は、mpirun コマンドでプログラムの実行を行います。

MPI プロセス数が、確保した VE の物理コア数（VE 数× 16）を超えると演算性能が著しく低下しますのでご注意ください。

逐次実行や自動並列化/OpenMP 並列実行用でコンパイルされたプログラムは、mpirun コマンドで実行を行っても複数 VE での実行はされません。

リスト 18　ジョブスクリプトの例（MPI 並列実行）

```
#!/bin/sh
#PBS -q sxs                     # AOBA-Sを使用する
#PBS --venode 8                 # VEを8個使う
#PBS -l elapstim_req=2:00:00    # 最大経過時間を2時間に指定

cd $PBS_O_WORKDIR               # qsubを実行したディレクトリに移動
mpirun -np 128 ./a.out          # カレントディレクトリの a.out を128プロセス並列で実行
```

■MPI と自動並列化/OpenMP 同時並列実行の場合　リスト 19 に MPI と自動並列化/OpenMP の同時並列実行の場合のジョブスクリプトの例を示します。

（MPI プロセス数）×（VE 内並列数）が、確保した VE の物理コア数（VE 数× 16）を超えると演算性能が著しく低下しますのでご注意ください。

リスト 19　ジョブスクリプトの例（MPI と自動並列化／ OpenMP 同時並列実行）

```
#!/bin/sh
#PBS -q sxs                     # AOBA-Sを使用する
#PBS --venode 8                 # VEを8個使う
#PBS -l elapstim_req=2:00:00    # 最大経過時間を2時間に指定
#PBS -v VE_OMP_NUM_THREADS=16   # VE内は16コア並列で実行

cd $PBS_O_WORKDIR               # qsubを実行したディレクトリに移動
mpirun -np 8 ./a.out          # カレントディレクトリの a.out を8プロセス×16コア並列で実行
```

■標準出力、標準エラーファイルをプロセス毎に分割出力する　標準出力および標準エラー出力は、1 つのファイルに各プロセスからの出力が混在して書き出されます。このとき、システムで用意されたシェルスクリプト mpisep.sh を利用すると、同一ファイルにプロセス毎の出力が入り混じって出力されないように、MPI プロセスごとにファイルを分けて出力することができます。リスト 20 に示したように、シェルスクリプト/opt/nec/ve/bin/mpisep.sh を実行形式ファイル a.out の前に記述し、環境変数 NMPI_SEPSELECT に 1 から 4 の値を指定することで、表 5 に示した動作を選択できます。

リスト 20　出力をプロセス毎に分割する方法

```
#!/bin/sh
#PBS -q sxs                     # AOBA-Sを使用する
#PBS --venode 8                 # VEを8個使う
#PBS -l elapstim_req=2:00:00    # 最大経過時間を2時間に指定
#PBS -v NMPI_SEPSELECT=3        # 出力形式3を指定

cd $PBS_O_WORKDIR               # qsubを実行したディレクトリに移動
mpirun -np 128 /opt/nec/ve/bin/mpisep.sh ./a.out
                                # カレントディレクトリの a.out を128プロセス並列で実行
```

表 5　NMPI_SEPSELECT の引数と出力形式

NMPI_SEPSELECT の引数	出力形式
1	MPI プロセスの標準出力をプロセス別のファイルに保存
2	（既定値）MPI プロセスの標準エラー出力を プロセス別のファイルに保存
3	MPI プロセスの標準出力および標準エラー出力を それぞれプロセス別のファイルに保存
4	MPI プロセスの標準出力および標準エラー出力を プロセス別の 1 つのファイルに保存

実行時に stdout.*や stderr.*の同名ファイルが存在する場合は、上書きではなくファイルに追記します。

4.4.5 qsub コマンドの主なオプション

表 6 に、qsub コマンドの主なオプションと機能を示します。ジョブスクリプトの冒頭から、#PBS の後に各オプションを記述します。

表 6 qsub コマンドの主なオプションと機能

オプション	機能	指定
-q 投入キュー名	投入キュー名を指定	必須
–venode 利用 VE 数	AOBA-S で利用する VE 数を指定 （このオプションは先頭のハイフンが 2 つ）	必須
-l elapstim_req=hh:mm:ss	最大経過時間を指定	必須（注 1 参照）
-A 課金先番号（PJ）	課金先の番号を指定	任意（注 2 参照） 省略時：デフォルトの PJ
-N リクエスト名	リクエスト名を指定	任意 省略時：ジョブスクリプトファイル名
-o ファイル名	標準出力のファイル名を指定	任意　省略時： リクエスト名.o リクエスト ID
-e ファイル名	標準エラー出力のファイル名を指定	任意　省略時： リクエスト名.e リクエスト ID
-jo	標準出力および標準エラー出力を 同一ファイルに出力	任意
-m b	リクエスト実行開始時にメールを送信	任意
-m e	リクエスト実行終了時にメールを送信	任意
-m be	リクエスト実行開始時と終了時にメールを送信	任意
-M メールアドレス	メールの送信先を指定	任意
-r y または n	リクエストのリラン可否を指定（注 3 参照）	任意　省略時：y

（注 1） 経過時間はバッチリクエストが実行を開始してから、終了するまでの時間です。プログラムの実行に、最大でどれくらいの経過時間の確保が必要かを指定します。指定した最大経過時間が短いリクエストほど計算資源が確保されやすく、実行待ちの時間を短縮することができます。

ただし、指定した最大経過時間を超えると、実行は打ちきりとなり強制終了します。I/O が高負荷となる場合、経過時間が想定よりも長くなることがありますので、必要十分な時間を指定してください。

（注 2） デフォルトの課金先番号や、利用可能な課金先番号の確認は、フロントエンドサーバ上で project コマンドを実行します。デフォルトの課金先番号の変更も可能です。

（注 3） リランとは、リクエストを始めから実行し直すことで、計算機のメンテナンスや障害発生時に管理者側でリクエストをリランする場合があります。

リランによりプログラムの実行に不都合が生じる場合（実時間で時間計測を行っている場合など）は、-r n（リランしない）を指定してください。

4.4.6 バッチリクエストの投入

バッチリクエストの投入は qsub コマンドで行います。リスト 21 に qsub コマンドの実行例を示します。ジョブスクリプトファイル名が run.sh の場合です。

リスト 21 qsub コマンドによるバッチリクエスト投入

```
(sfront)$ qsub run.sh
```

バッチリクエストが正常に投入されるとリスト 22 のようなメッセージが表示されます。この例では、リクエスト ID には 1234.sjob が割り当てられ、投入先は sxs キュー、課金先番号は un0000 が指定されたことを示しています。リクエスト ID はバッチリクエストの状況確認やキャンセルの際に用います。

バッチリクエストの投入が失敗した場合は、エラーメッセージが表示されますので、ジョブスクリプトの記述などに誤りがないかを確認してください。

リスト 22　qsub コマンドによるバッチリクエスト投入

```
(sfront)$ qsub run.sh
プロジェクトコード:un0000 にリクエストを投入します
Request 1234.sjob submitted to queue : sxs.
```

4.4.7　バッチリクエストの実行

投入されたリクエストは、実行待ちの列に並びます。リクエストの実行順序はバックフィルスケージューリングで制御されます。

バックフィルスケージューリングでは、計算資源が確保できたリクエストから実行を開始します。実行に必要な計算資源量（利用 VE 数×最大経過時間）が少ないリクエストは、投入順序によらず早く実行開始する場合があります。

4.4.8　バッチリクエストの状態確認

バッチリクエストの状態確認は reqstat コマンドで行います。reqstat コマンドを実行すると、実行待ちおよび実行中のリクエストの状態が表示されます。該当するリクエストがない場合は何も表示されません。

表 7 に reqstat コマンドの表示項目を、表 8 にリクエストの状態の説明を示します。

表 7　reqstat コマンドの表示項目

表示項目	内容
RequestID	リクエスト ID
RequestName	リクエスト名
User	利用者番号
PJCode	課金先番号（プロジェクトコード）
Que	実行キュー名
Node	実行時に指定した VE 数
ElapseLimit	投入時に指定した最大経過時間
STT	リクエストの状態
StartTime	実行開始予定時刻 （実行開始後は実際の実行開始時刻）
Memory1	現在の VH 使用メモリ量
Memory2	現在の VE 使用メモリ量
ElapseTime	現在までの経過時間
NodeTime	現在までのノード時間（ElapseTime × Node） （課金対象時間）

表 8　reqstat コマンドの表示項目

表記	リクエストの状況
RUN	実行中
QUE	実行待ち
EXT	標準出力/標準エラー出力ファイルの出力中（※）

リクエストの実行開始予定時刻は、StartTime の欄に表示されます。実行開始予定時刻は他のリクエストの状況により随時更新されます。通常は更新前の時刻よりも遅くなることはありませんが、システムの障害発生時にはその限りではありません。
※　以下のような場合、リクエストが EXT 状態で長く停滞することがあります。他の利用者のリクエストの実行に影響が出ますのでご注意ください。

- ファイル容量がクォータ値を超えたため、標準出力/標準エラー出力ファイルを出力できない
  ファイルを整理して容量を空けてください。空き容量が確保されるまで EXT 状態で停滞したままになります。容量が確保され次第、ファイルが出力されリクエストが終了します。
- システム上で問題が発生している
  管理者で対応しますのでそのままお待ちください。状況をメールでご連絡いたします。

ファイル容量の追加方法は以下のページをご参照ください。
【データ転送（ストレージ）】https://www.ss.cc.tohoku.ac.jp/storage/

4.4.9　バッチリクエストのキャンセル

投入したリクエストをキャンセルする場合は、qdel コマンドを使用します。qdel コマンドを実行すると実行中のプログラムは強制終了し、リクエストは削除されます。リクエストのキャンセルは投入した利用者番号から行うことができ、他利用者のリクエストはキャンセルできません。

リスト 23 に qdel コマンドの実行例を示します。1234.sjob というジョブ ID のリクエストをキャンセルする例で、qdel に続けてキャンセルするリクエスト ID を指定します。リクエスト投入時、または reqstat コマンドで表示されるリクエスト ID を指定してください。

リクエストがキャンセルされるとメッセージが表示されます。

リスト 23　qdel コマンド

```
(sfront)$ qdel 1234.sjob
Request 1234.sjob was deleted.
```

4.4.10　バッチリクエストの終了

リクエストが終了すると、reqstat コマンドで表示されなくなります。終了したリクエストの実行結果ファイルとして、リクエスト実行中に標準出力された内容を保存した「標準出力ファイル」と、標準エラー出力の内容を保存した「標準エラー出力ファイル」が作成されます。

ジョブのスクリプトでオプション -o と-e を指定した場合はそのファイル名で作成されます。指定しない場合は以下のファイル名で作成されます。

- 標準出力ファイル：　リクエスト名.o リクエスト ID
- 標準エラーファイル：　リクエスト名.e リクエスト ID

なお、標準出力/標準エラー出力のファイルサイズの上限値は 1GB です。ファイルサイズが上限値を超えた場合はプログラムが強制終了しますので、実行結果はプログラム内でファイル名を指定して書き

出すようにしてください。

4.5　会話リクエストの投入と利用

AOBA-S では、qlogin コマンドを利用したセッション接続タイプの会話リクエストの投入が可能です。会話リクエストでは 1VH+8VE を利用して、ソースコードのコンパイル、会話型形式での実行、および GDB（ve-gdb）の利用が可能です。

会話リクエストはバッチリクエストと同様に演算負担額が発生します。会話リクエストのセッションが開始されてからセッションが切断されるまで、が演算負担額の対象となります。ジョブの実行を行っていない場合でも、セッションの接続時間が課金対象時間となりますのでご注意ください。

会話リクエストの最大経過時間は 1 時間です。会話リクエスト用ノードの利用状況により、セッションの開始まで実行待ちが発生することがあります。

4.5.1　qlogin による会話リクエストの投入方法

会話リクエストの投入は、AOBA-S 用フロントエンドサーバ上で qlogin コマンドで行います。リスト 24 に qlogin コマンドの例を示します。

リスト 24　qlogin コマンド

```
(sfront)$ qlogin -q inter -l elapstim_req=1800
                              # 会話キューinterの指定（必須）と最大経過時間の指定（任意）
プロジェクトコード：un0000 にリクエストを投入します
Request 1234.sjob submitted to que: inter. # 会話リクエストが受け付けられた
Wating for 1234.sjob to start.             # 実行ホストとのセッション接続待ち
(sxat3001)$                                # 実行ホストのシェルプロンプトに表示が変わる
(sxat3001)$ exit                           # セッションの切断
(sfront)$                                  # シェルプロンプト表示が戻る
```

リスト 25 に ve-gdb（VE 用 GDB）の起動方法を示します。

リスト 25　ve-gdb コマンド

```
(sxat3001)$ ve-gdb a.out
No symbol table is loaded.  Use the "file" command.
GNU gdb (GDB) 7.12.1-8.el8
Modified by NEC Corporation for the VE port, 2017-2019
Modified by Arm. Copyright (C) 2002-2019 Arm Limited (or its affiliates).
All rights reserved.
（省略）
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...(no debugging symbols found)...done.
(gdb)
```

VE 用の GDB と一般的な GDB の相違点については、6 章の GDB の相違点をご参照ください。

5 SX-Aurora TSUBASA 用数値演算ライブラリ

5.1 NLC (NEC Numeric Library Collection)

NEC Numeric Library Collection は、広範な分野の数値シミュレーションプログラムの作成を強力に支援する数学ライブラリのコレクションであり、Vector Engine に対応しています。NEC Numeric Library Collection を用いることにより、難解な数値計算アルゴリズムの詳細に煩わされることなく高度な科学技術計算プログラムを作成することができ、数値シミュレーションプログラム開発の生産性を大幅に改善することができます。

NEC Numeric Library Collection は、Fortran または C 言語プログラムから利用できます。

5.1.1 Fortran プログラムから利用する場合

Fortran プログラムから利用する場合、表 9 に示すライブラリが使用できます。

表 9 Fortran 用 NLC の機能概要

ライブラリ名		機能概要
ASL	ネイティブインタフェース	数値計算・統計計算のための各種アルゴリズムを備えた 科学技術計算ライブラリ
	統合インタフェース	フーリエ変換、乱数、ソート
	FFTW3 インタフェース	FFTW (version 3.x) の API で ASL のフーリエ変換を 利用するためのインタフェースライブラリ
BLAS		ベクトル、行列の基本演算
LAPACK		連立 1 次方程式、固有値方程式、特異値分解
ScaLAPACK		連立 1 次方程式、固有値方程式、特異値分解 (分散メモリ並列用)
BLACS		ベクトル、行列の基本演算のためのメッセージパッシングライブラリ (分散メモリ並列用)
SBLAS		スパース行列の基本演算
HeteroSolver		連立 1 次方程式 (スパース行列用の直接法ソルバ)
Stencil Code Accelerator		ステンシル計算の加速

5.1.2 C プログラムから利用する場合

C プログラムから利用する場合、表 10 に示すライブラリが使用できます。

表 10 C 言語用 NLC の機能概要

ライブラリ名		機能概要
ASL	ネイティブインタフェース	数値計算・統計計算のための各種アルゴリズムを備えた 科学技術計算ライブラリ
	統合インタフェース	フーリエ変換、乱数、ソート
	FFTW3 インタフェース	FFTW (version 3.x) の API で ASL のフーリエ変換を 利用するためのインタフェースライブラリ
CBLAS		BLAS C 言語インタフェース
SBLAS		スパース行列の基本演算
HeteroSolver		連立 1 次方程式 (スパース行列用の直接法ソルバ)
Stencil Code Accelerator		ステンシル計算の加速

NLC の詳細、およびコンパイルとリンク方法については 6 章の NLC (NEC Numeric Library Collection) ユーザーズガイドをご参照ください。

6　マニュアル

サブシステム AOBA-S についてのマニュアルはオンライン上で公開されています。以下リンク先の NEC Aurora Forum の NEC SX-Aurora TSUBASA Documentation をご参照ください。英語版、日本語版ともに提供されています。

【NEC Aurora Forum Documentation】https://www.hpc.nec/documentation

当センター独自の運用方法により、マニュアルに記載の事項が動作しない場合もありますのでご注意ください。

6.1　コンパイラマニュアル

コンパイラについては以下のマニュアルをご参照ください。

■SDK

- C/C++ Compiler ユーザーズガイド、C/C++ Compiler User's Guide
- Fortran Compiler ユーザーズガイド、Fortran Compiler User's Guide
- NEC Parallel Debugger ユーザーズガイド、NEC Parallel Debugger User's Guide

■NEC MPI

- NEC MPI ユーザーズガイド、NEC MPI User's Guide

6.2　性能情報取得についてのマニュアル

実行時の性能情報取得については以下のマニュアルをご参照ください。

■SDK

- PROGINF/FTRACE ユーザーズガイド、PROGINF/FTRACE User's Guide
- NEC Ftrace Viewer ユーザーズガイド、NEC Ftrace Viewer User's Guide

6.3　NQSV についてのマニュアル

NQSV の利用方法については以下のマニュアルをご参照ください。

■NQSV

- NQSV 利用の手引 操作編、NQSV User's Guide Operation

6.4　GDB についてのマニュアル

VE 用の GDB と一般的な Linux の GDB の使用法に関する相違点については以下の資料をご参照ください。

■VEOS

- GDB の相違点

6.5 数値計算ライブラリ（NLC）マニュアル

数値計算ライブラリ（NLC）については以下のマニュアルをご参照ください。

■SDK

- NLC (NEC Numeric Library Collection) ユーザーズガイド、
  NLC (NEC Numeric Library Collection) User's Guide

7 おわりに

本稿では、スーパーコンピュータ AOBA のサブシステム AOBA-S のプログラミング利用ガイドとして基本的な手順を紹介しました。研究室のサーバでは実現できなかったプログラムやアイデアを、ぜひ最新鋭のスーパーコンピュータでお試しください。研究の強力なツールとしてご活用いただければ幸いです。

ご不明な点、ご質問等がありましたら、お気軽にセンターまでお問い合わせください。お問い合わせについては利用相談フォームをご利用ください。

【利用相談フォーム】https://www.ss.cc.tohoku.ac.jp/consultation/

センターから運用に関するお知らせは以下をご参照ください。

【センターからのお知らせ】https://www.ss.cc.tohoku.ac.jp/information/

［ 共同研究成果 ］

後退平板境界層における横流れ渦と主流乱れの相互作用による乱流遷移の直接数値解析

中川 皓介[1]・塚原 隆裕[2]
1：東京理科大学大学院 創域理工学研究科 機械航空宇宙工学専攻
2：東京理科大学 創域理工学部 機械航空宇宙工学科

航空機の巡航速度高速化に伴って後退翼が標準採用されるようになり，衝撃波を抑制することで性能向上が図られてきた．しかし，後退翼面上では主流方向と圧力勾配方向の違いから流れ場は 3 次元境界層を形成し，横流れ不安定を引き起こす．この横流れ不安定は翼前縁近傍での乱流遷移を促し，乱流摩擦抵抗が増加することで，結果的に燃費性能の悪化に繋がる．横流れ不安定による乱流遷移現象の解明及び制御は，航空機開発に関わる流体力学の課題の 1 つである．私共は，直接数値解析を用いて主流乱れと壁面粗さによる各変動の相互作用による乱流遷移過程を調査している．本報では，既報の原著論文から抜粋して，メカニズム解明の進んだ横流れ渦と主流乱れの相互作用について概説する．ここでは，短波長主流乱れは強制遷移を促進する一方，長波長主流乱れは粗さ要素近傍の強い変動を抑制し，強制遷移を抑えることが明らかとなった．

1.　序論

航空分野における研究開発は今日まで飛躍的な進歩を遂げている．その中で流体力学分野に着目すると，翼面上の流れが層流から乱流へ遷移するメカニズムの解明及び制御，遷移位置の予測などに対して注力されている[1]．なぜならば航空機における流動抵抗は壁面摩擦が多くを占め，さらに乱流の摩擦抵抗は層流の数倍以上（または桁違いに）大きくなるためである．層流―乱流遷移位置の正確な把握や予測は，摩擦抵抗の定量的評価に直結しており，航空機設計において重要な指針となる．乱流遷移機構の解明を目指すにあたって，その研究方法は実機による飛行試験，縮小モデルなどによる風洞試験，シミュレーションによる数値計算が挙げられる．実機による飛行試験は理想的な手段であるが，費用面や安全面から多くの場合は風洞試験や数値計算によって性能評価が事前に行われる．しかし，一般的な風洞試験では Re/Ma（レイノルズ数/マッハ数）が実機に比べて低くなり，乱流化が過小評価される場合がある．そこで孤立円柱を粗さ要素とした粗面によって乱流化を誘発したうえで風洞試験が行われる．また粗さには渦を故意に誘起させ，その乱流遷移過程を調査する目的でも使用され[2–4]，孤立粗さ要素は典型的試験方法の 1 つである．

また実験における環境，特に風洞試験機にはその性質上，主流乱れと呼ばれる非定常の撹乱を主流に含む．この主流乱れは，低強度ながら飛行環境でも観察される．主流乱れは翼面上や試験部分の流れ場に影響を与えることで乱流遷移過程を変化させることが明らかになっている[2, 5]．また実試験において主流乱れは主に主流速度に対する変動速度の割合で評価されるのが一般的である[2, 4]．これは熱線流速計などの計測機器で主流方向速度を計測することは容易であるが，3 次元的な計測は技術，費用及び必要性の観点から実施されない場合がある．そのため実験を通して主流乱れが与える影響を実験的研究から議論する際には乱れ強さのみで議論が行われる[2]ことが多く，主流乱れを形成する変動の時空間分布や多方向の変動による影響を加味した議論は乏しい．そこで本研究では直接数値計算において円柱粗さと主流乱れを模擬した解析を行うことで，風洞試験時や定常飛行時においても外的要因によって時々刻々と変化する主流乱れのパラメータを一定に制御し，流れ場の影響及び乱流遷移への影響を評価する．

航空機の翼についても，航空分野の発展に伴い様々な翼平面形が考案，研究そして開発されて

きた．その中で本研究が対象とするのは後退翼である．後退翼採用の背景には航空機の性能向上に伴い巡航速度が高速化するにつれて，翼面上の負圧面で加速された流体が音速に近づき衝撃波が生じる点について問題となったことが契機である．衝撃波は揚力の低下や翼面上の流れ場の変化をもたらし，本来の性能を低下させる抵抗増大に繋がる．その衝撃波抑制を目的として，機体に対して主翼を後方に傾けた後退角を有する後退翼が開発され採用されるようになった．しかし，後退翼は主流方向に対して傾いた圧力勾配を有することになり，後退翼面上の流れ場はいわゆる捻れた 3 次元境界層（図 1）を形成することになる[1]．この特徴的な境界層内で生じる後退翼特有の流体現象が横流れ不安定である．横流れ不安定は後退翼前縁で支配的であり乱流遷移をもたらす[6]．また横流れ不安定は薄い境界層中で生じ，壁面の僅かな粗さや主流乱れに対しても敏感に反応すると明らかになっている．そこで本研究の意義として，直接数値解析によって主流乱れと壁面粗さによってもたらされる変動の相互作用による乱流遷移過程を調査し，乱流遷移過程の解明を目指すものである．

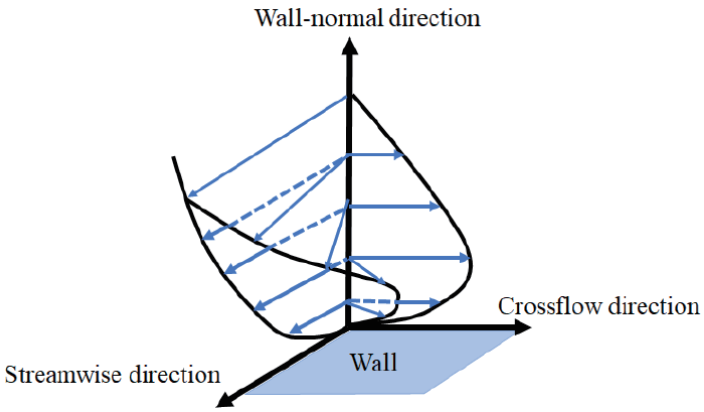


図 1 後退平板境界層模式図

2. 計算手法

本研究の解析対象は，横流れ不安定が支配的である後退翼前縁の流れ場を平板で模擬した後退平板と呼ばれる系である．後退平板上の基本流を再現するため，Falkner–Skan–Cooke(FSC)相似解を採用した．解析対象を図 2 に示す．本研究ではコード（翼弦）方向を x，スパン方向を y，壁面垂直方向を z としている．FSC 相似解を基本流として，その乱流遷移過程を直接数値解析（DNS: direct numerical simulation）で調査する．

流体の速度と圧力に関する支配方程式は，非圧縮性の連続の式と Navier-Stokes 方程式

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \frac{1}{\mathrm{Re}} \nabla^2 \mathbf{u}. \tag{2}$$

である．空間的離散化にはコード方向とスパン方向に 4 次精度中心差分，壁面垂直方向に 2 次精度中心差分を用いている．境界条件は，流入境界に FSC 相似解，流出条件に対流流出，壁面に滑り無し条件，そして壁面垂直方向遠方に FSC 相似解としている．数値安定のため，流出境界には流れ場の速度を基本流である FSC 相似解に漸近させるフリンジ領域（厚さを L_f とする）を設けている（図 2）．計算条件を表 1 に記す．式(2)および表 1 におけるレイノルズ数は流入部遠方コード方向速度 U_0 及び流入部排除厚さ δ_0^*で定義され，本研究における全ての計算条件に関して統一されている．また，U_0 及び δ_0^*は無次元化にも用いる．m, φ_0 はそれぞれ FSC 相似解における加速係数と流入部の流線曲がり角であり，先行研究[7–9]と一致している．表 2 に示す計算領域は，横流れ渦の遷移過程を捉えるため，横流れ渦の最不安定波長と一致している．

表 1　計算条件

Domain	Re	m	φ_0	$L_{fx}\times L_{fz}$	$L_x\times L_y\times L_z$	$N_x\times N_y\times N_z$
Short	337.9	0.34207	55.3°	$40\delta_0^*\times 4\delta_0^*$	$200\delta_0^*\times 25.14\delta_0^*\times 27\delta_0^*$	1024×128×128
Long	337.9	0.34207	55.3°	$40\delta_0^*\times 4\delta_0^*$	$400\delta_0^*\times 25.14\delta_0^*\times 27\delta_0^*$	2048×128×128

表 2　円柱粗さ高さ及び粗さレイノルズ数

k_z/δ_0^*	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0	2.1
Re_{kk}	342	401	491	538	583	666	717	810	867	967	1030

表 3 初期撹乱計算条件

Tu	$\lambda_{\max}/\delta_0^*$	$L_{dx}\times L_{dy}\times L_{dz}$	$N_{dx}\times N_{dy}\times N_{dz}$
0.01% – 0.5%	5.0 – 25.14	$80\delta_0^*\times 25.14\delta_0^*\times 27\delta_0^*$	$4096\times 128\times 128$

後退平板上に模擬する円柱粗さは，埋め込み境界法を用いて仮想の体積力を付与することで DNS において再現する．円柱の直径は $d/\delta_0^*=6.0$，円柱中心の位置は $x/\delta_0^*=20.59$ である．ここで粗さのスパン方向間隔は $\lambda/\delta_0^*=25.14$ であるが，これは横流れ渦の最不安定波長と一致している[7, 8]．また，本研究では粗さ高さに関するパラメトリックスタディを行う．円柱粗さに関する条件を表 2 にまとめる．また先行研究[8, 9, 11]に倣って，円柱粗さ高さと円柱頂点位置における FSC 相似解の基本流$u_k=(u_{\mathrm{FSC}}^2(k_z)+v_{\mathrm{FSC}}^2(k_z))^{0.5}$をそれぞれ代表速度と代表長さにした粗さレイノルズ数 $Re_{kk}=k_zu_k/\nu$ を合わせて表記する．

主流乱れが乱流遷移に与える影響調査のため，本研究では主流乱れを模した撹乱を計算領域に流入させる．この手法は Watanabe and Maekawa[12]の研究を参考にした．この手法では

$$E(k)=k^4\exp\left(-\frac{2k}{k_{\max}}\right) \tag{3}$$

に基づいて任意波数 $k_{\max}$ にピークを有するエネルギースペクトル分布を規定し，得られたスペクトル分布を波数空間から物理空間へ逆フーリエ変換を行うことで初期撹乱を得る．物理空間の撹乱の乱れ強さ Tu は，各方向の変動速度を用いて

$$Tu=\frac{1}{U_0}\sqrt{\frac{1}{3}\frac{\int\int\int(u'^2+v'^2+w'^2)\mathrm{d}x\mathrm{d}y\mathrm{d}z}{L_{dx}L_{dy}L_{dz}}}\times 100\% \tag{4}$$

と定義する．一様かつ等方的な初期撹乱には，計算領域へ流入させる際に壁面滑り無し条件を満たすよう，FSC 相似解の速度プロファイルと一致した窓関数が適用される．主流乱れを模擬した初期撹乱に関する計算条件を表 3 に示す．また初期撹乱領域における格子数は表 1 に示した後退平板領域の条件と一致している．

本計算の実行には，主に東北大学サイバーサイエンスセンター所有のスーパーコンピュータ群 AOBA を利用した．ベクトル型スーパーコンピュータ SX-Aurora TSUBASA を用いて，MPI および OpenMP 並列による自作 Fortran コードを用いて DNS 及び Matlab によるデータ処理を実施している．本研究の解析条件の 1 つとして，格子数 $N_x\times N_y\times N_z=2048\times 128\times 128$ で 10000 ステップ分の 1000 ステップ毎の 3 次元速度場出力を含むプログラム性能を表 4 に示す．検証における 10 回分の出力においても速度場の出力形式に依存して実行時間に差異が生じ，バイナリ形式での出力では実行時間の 10%削減に成功している．

表 4　プログラム性能と出力形式の比較.
1000 ステップ毎の出力を含む 10000 ステップ分の計算時間.

	アスキー形式	バイナリ形式
実行時間 [sec]	5162.076	4626.460
CPU 時間 [sec]	20305.265	18143.728
GFlops 値 (real)	63.292	70.616
ベクトル演算率 [%]	90.520	90.956
メモリサイズ [GB]	44.842	44.873

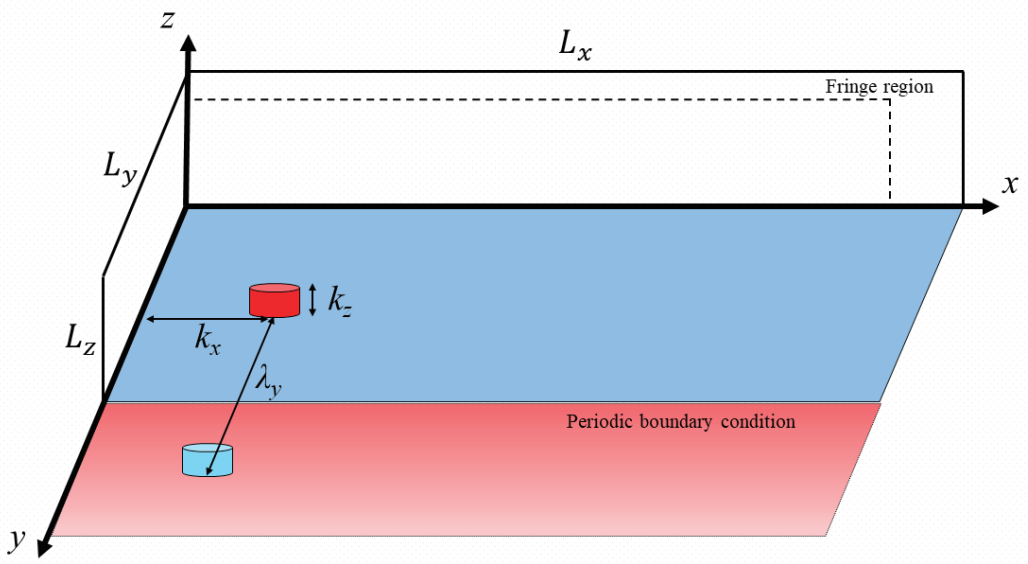


図 2 解析対象．底面の青い部分が DNS 計算領域．周期境界条件により，z 方向に円柱粗さ要素（高さ k_z）が等間隔（間隔 λ_y）に配列している．

3.　結果

本章の結果は既報の学術論文[10]に基づく概要である．後退平板境界層乱流遷移過程における，円柱粗さで誘起される定在波と主流乱れの相互作用による乱流遷移過程を示すにあたり，先ず主流乱れのない条件における横流れ渦の乱流遷移過程を示す．

図 3 に統計量として壁面摩擦係数 C_f のコード方向分布を示す．粗さが高くなるにつれて摩擦係数は増加する．このとき流れ場には円柱後流に馬蹄渦を含む定在波が形成され，下流にかけて成長し横流れ渦と呼ばれる後退平板で生じる特徴的な縦渦を形成する．粗さ高さが $k_z/\delta_0^* = 1.7$ では摩擦係数の断続的上昇が確認できる．このとき流れ場は乱流化しており，$k_z/\delta_0^* \leq 1.6$ で生じる定在波は形成されない．このように摩擦係数分布の変化に基づいて遷移過程を 2 種類に大別することができ，定在波及び横流れ渦が形成されず乱流遷移する過程を「強制遷移」と呼称する．そして円柱回りの定在波が横流れ渦を形成し，やがて乱流遷移する過程は，段階を踏む遷移過程であるので「段階遷移」と呼称する．この 2 種類の遷移形態からそれぞれ代表して，$k_z/\delta_0^* = 1.7$ と $k_z/\delta_0^* = 1.3$ の粗さ高さ条件における流れ場の可視化を図 4 に示す．図 4(a)は強制遷移の流れ場であり，円柱を起点にヘアピン渦が形成されている．先行研究においても同様に，高い円柱粗さ高さによって，円柱背後の定在波にヘアピン渦が生じることが報告されており，ヘアピン渦による定在波破壊と乱流遷移が強制遷移の特徴である[8, 11]．一方で段階遷移の流れ場を示した図 4(b)では，円柱の周りに生じた渦構造が下流にかけて成長し，やがて一次構造としての横流れ渦を形成する乱流

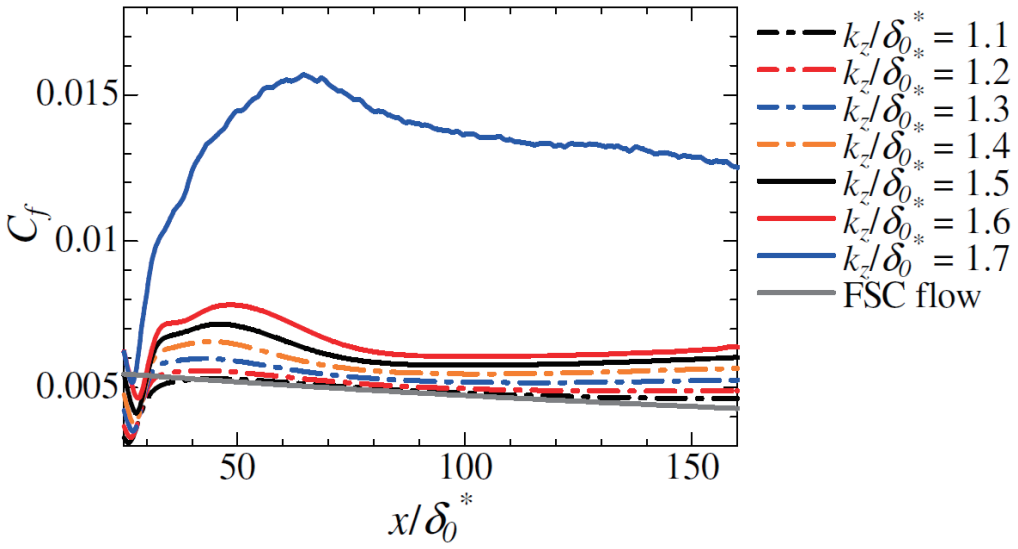


図 3　主流乱れのない条件 $Tu = 0\%$における摩擦係数分布の粗さ高さ依存性．灰色線は FSC 相似則（層流）．[K. Nakagawa, T. Tsukahara, and T. Ishida, 2023, Aerospace, 10(2), 128; licensed under a CC-BY license. Copyright © 2023 The Authors. Published by MDPI, Basel, Switzerland.]

遷移の段階が可視化される．形成された横流れ渦は $x/\delta_0^* = 200$ 付近で，二次的な構造の発生と，それによる一次構造の崩壊が開始される．この二次渦構造の拡大図では，一次構造である横流れ渦に指がかかるような渦構造であることが分かる．この視覚的特徴から二次渦構造は finger(-like) vortex と呼称されている[8, 13-15]．遷移過程の種類が変わる条件を臨界粗さレイノルズ数と呼び，その値は表 2 から $\mathrm{Re}_{kk} = 666$–717 である．

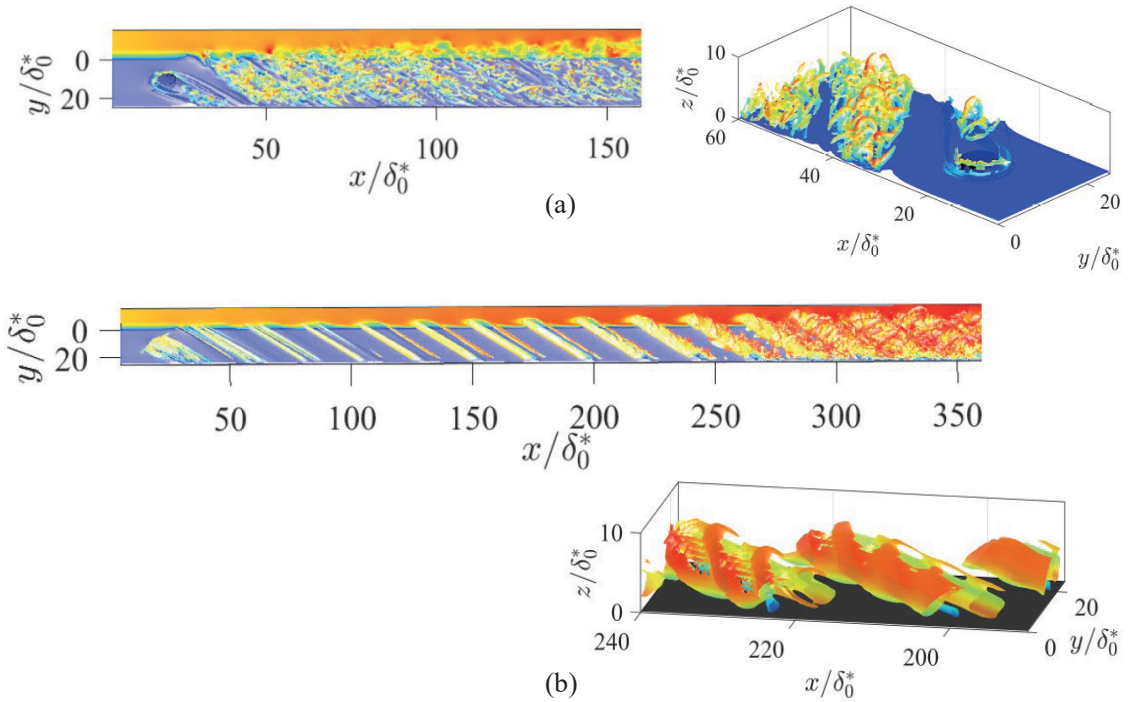


図 4　流れ場の 3 次元可視化．(a) 強制遷移，(b) 段階遷移．等値面は Q 値の等値面（コード方向速度による色づけで，赤いほど高速を示す）と低速領域 $u/U_0 = 0.3$ の等値面（青）．[K. Nakagawa, T. Tsukahara, and T. Ishida, 2023, Aerospace, 10(2), 128; licensed under a CC-BY license. Copyright © 2023 The Authors. Published by MDPI, Basel, Switzerland.]

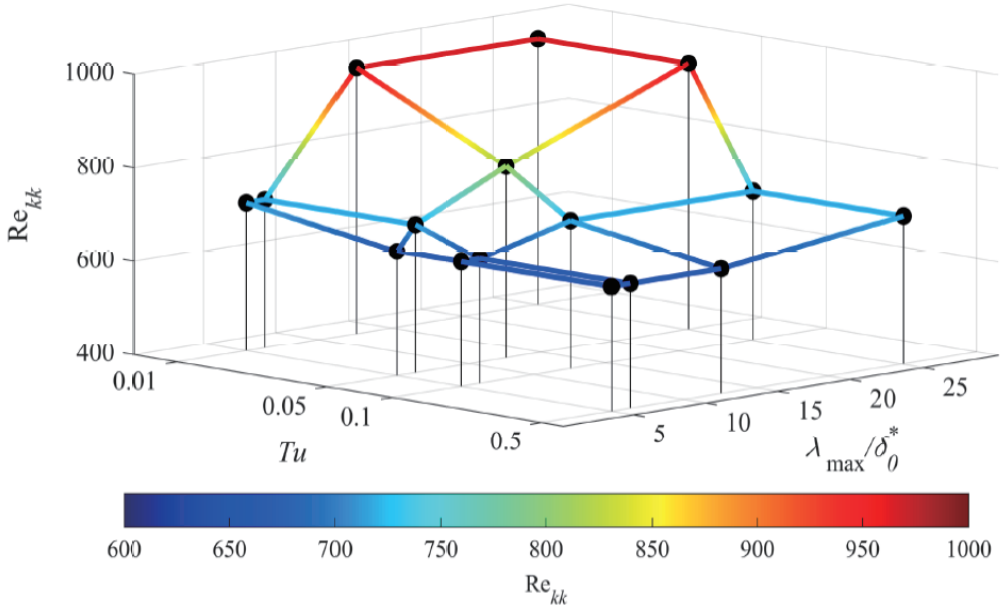


図 5　臨界粗さレイノルズ数 Re_{kk} における主流乱れ（乱れ強さ Tu とピーク波長 λ_{max}）依存性．[K. Nakagawa, T. Tsukahara, and T. Ishida, 2023, Aerospace, 10(2), 128; licensed under a CC-BY license. Copyright © 2023 The Authors. Published by MDPI, Basel, Switzerland.]

3.1　臨界粗さレイノルズ数に対する主流乱れの影響

図 5 に臨界粗さレイノルズ数に関する主流乱れの乱れ強さと波長に関するパラメトリックスタディの結果を示す．主流乱れの波長に依存しない共通した傾向として，乱れ強さの増加は臨界粗さレイノルズ数の低下をもたらす．強い主流乱れにさらされた流れ場はより崩壊しやすく，主流乱れのない条件と比較して強制遷移となりやすい[5]．主流乱れ波長依存性について，高主流乱れ $Tu \geq 0.1\%$では波長ごとの臨界値の変化がほとんど現われない．一方で低主流乱れ強さ条件では，短波長主流乱れと長波長主流乱れを比較したとき臨界値が大きく異なる．特に長波長主流乱れで

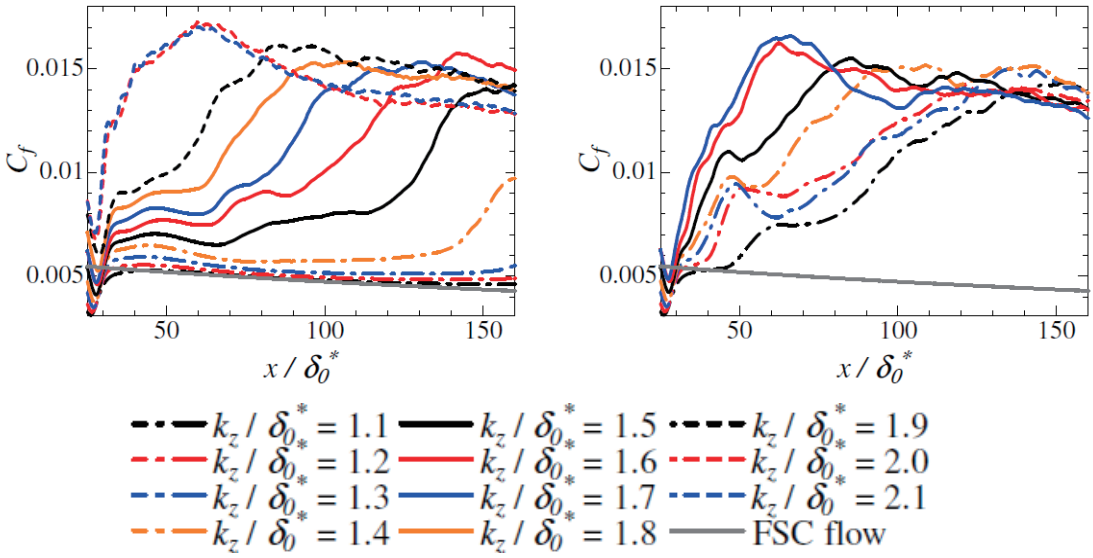


図 6　$Tu = 0.05\%$における摩擦係数分布の粗さ高さ依存性．左）長波長主流乱れ条件，右）短波長主流乱れ条件．[K. Nakagawa, T. Tsukahara, and T. Ishida, 2023, Aerospace, 10(2), 128; licensed under a CC-BY license. Copyright © 2023 The Authors. Published by MDPI, Basel, Switzerland.]

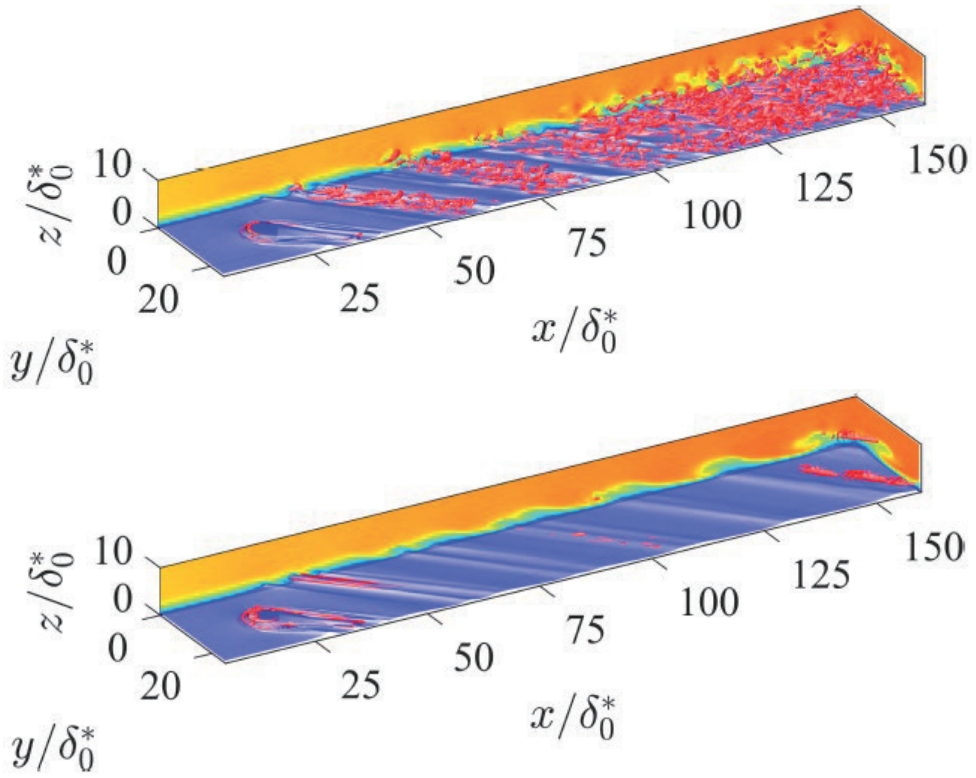


図 7　流れ場の 3 次元可視化．等値面は Q 値（赤）と低速領域 u/U_0=0.3（青）．　(上図): λ_{max}/δ_0^* = 5.0, (下図): λ_{max}/δ_0^* = 25.14. [K. Nakagawa, T. Tsukahara, and T. Ishida, 2023, Aerospace, 10(2), 128; licensed under a CC-BY license. Copyright © 2023 The Authors. Published by MDPI, Basel, Switzerland.]

は主流乱れのない Tu = 0%条件よりも高い臨界値となることから，強制遷移を抑制するメカニズムを有している．

3.2　段階遷移における主流乱れ波長依存性

図 5 に示した主流乱れと粗さ高さに関するパラメトリックスタディによって短波長主流乱れは強制遷移を促進，長波長主流乱れは強制遷移を抑制することが明らかになった．そこで本節では短波長主流乱れは λ_{max}/δ_0^* = 5.0 を対象に，長波長主流乱れは λ_{max}/δ_0^* = 25.14 を対象に，それぞれの摩擦係数分布を図 6 に示す．長波長主流乱れに関して，粗さ高さ $k_z/\delta_0^* \leq 1.3$ では摩擦係数が計算領域内で低い値を維持している．更に高い粗さ高さでは計算領域内での摩擦係数増加及び乱流遷移を確認できる．粗さ高さ $k_z/\delta_0^* < 1.7$ では遷移位置が主流乱れのない条件よりも上流である．つまり，長波長主流乱れは，強制遷移を抑制する効果が見られるが，臨界粗さ高さ以下の条件では乱流遷移を促進する．一方で粗さ高さ $k_z/\delta_0^* = 1.7$ では遷移位置が主流乱れのない条件よりも下流であり，遷移抑制の効果が摩擦係数分布からも見て取れる．短波長主流乱れ条件では，粗さ高さ $k_z/\delta_0^* \geq 1.1$ 全条件で Tu = 0%条件よりも上流での乱流遷移を示す．短波長主流乱れは粗さ高さに依らず乱流遷移を促進する．

それぞれ主流乱れの影響を受けたときの流れ場を可視化し，図 7 に示す．短波長主流乱れにさらされた流れ場は円柱後流に生じる低速領域に沿ってヘアピン渦を伴い，それらによって定在波が崩壊している．ヘアピン渦の発生とそれによる乱流遷移は強制遷移の特徴として述べた．ただし，摩擦係数分布から判断される遷移位置は強制遷移条件よりも下流であることから，短波長主

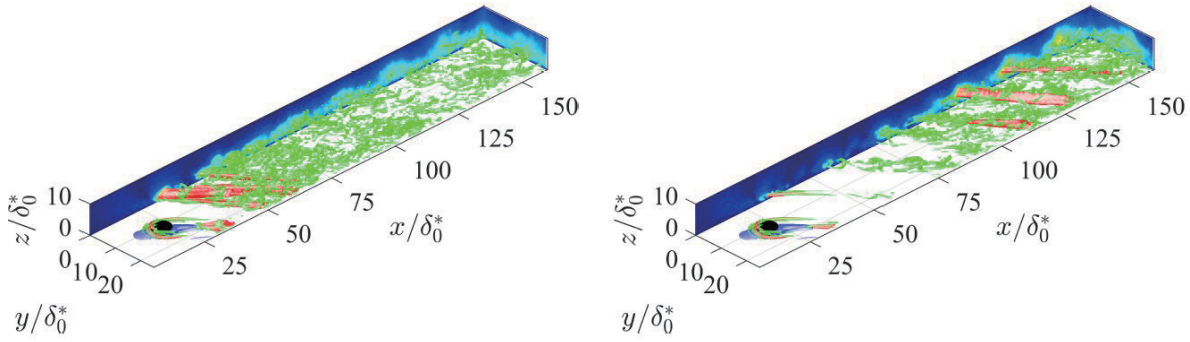


図 8　流れ場の 3 次元可視化．等値面は Q 値（緑），再循環領域（青）と壁面垂直方向変動速度 RMS 値 $w_{RMS}/U_0 = 0.12$（赤）．左図：短波長主流乱れ条件，右図：長波長主流乱れ条件．[K. Nakagawa, T. Tsukahara, and T. Ishida, 2023, Aerospace, 10(2), 128; licensed under a CC-BY license. Copyright © 2023 The Authors. Published by MDPI, Basel, Switzerland.]

流乱れにさらされた臨界粗さ高さを下回る条件における乱流遷移過程は強制遷移に準ずる傾向にある．長波長主流乱れではヘアピン渦の誘起は確認できない．計算領域の最下流で僅かに乱れた構造が生じるに留まっている．そのため，長波長主流乱れかつ低粗さ高さの条件における遷移過程は，横流れ渦の形成と成長を伴う段階遷移である．

3.3　強制遷移における主流乱れ波長依存性

図 5 に示すよう長波長主流乱れは強制遷移の臨界粗さレイノルズ数を Tu = 0%条件より引き上げることが分かった．そこで粗さ高さ $k_z/\delta_0^* = 1.7$ における流れ場と，円柱近傍での周波数解析結果を図 8, 9 に示す．短波長主流乱れ条件下では，円柱近傍からヘアピン渦が誘起され，ヘアピン渦の回転運動に伴って壁面垂直方向変動がもたらされる（図 8）．ヘアピン渦の回転運動については，2 次元境界層の壁乱流で生じるものと同様であった[16, 17]．ヘアピン渦による乱流遷移は図 4(a) に示した Tu = 0%条件と同様である．一方で，長波長主流乱れ条件では，円柱近傍の渦構造は一度減衰を経た後に，再度発生し乱流遷移に至る．流れ場の観察から，長波長主流乱れは円柱近傍で

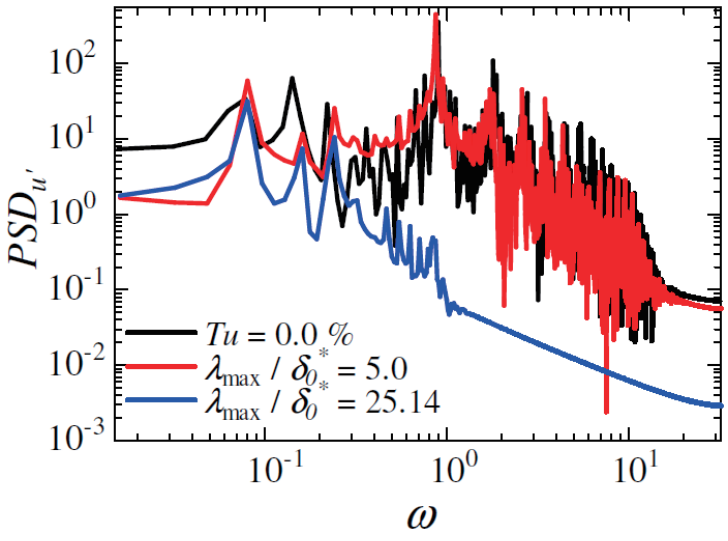


図 9　円柱近傍のコード方向変動速度に関する周波数解析結果．[K. Nakagawa, T. Tsukahara, and T. Ishida, 2023, Aerospace, 10(2), 128; licensed under a CC-BY license. Copyright © 2023 The Authors. Published by MDPI, Basel, Switzerland.]

発生する高周波変動を抑制することが示唆される．この減衰は周波数解析においても特徴として現われる．図 9 に示すように，強制遷移では $\omega \approx 0.9$ の高周波変動がピークを示している．横流れ渦乱流遷移過程において高周波変動は乱流遷移自体の引き金であると知られている．その高周波変動を長波長主流乱れによって打ち消すことが強制遷移抑制のメカニズムである．

4.　結論

本研究では後退平板を模擬した FSC 相似解を基本流とした流れ場に対して DNS を実施し，乱流遷移過程を調査した．その中でも主流乱れ強さ，波長，及び円柱粗さ高さに対してのパラメトリックスタディを実施し，遷移過程の分析と遷移形態の分類を行った．詳細は既報論文[10]を参照されたい．

謝辞

本研究は、東北大学サイバーサイエンスセンターのスーパーコンピュータを利用することで実現することができた．また，筆頭著者は JST 次世代研究者挑戦的研究プログラム JPMJSP2151 の支援を受けたものである．

参考文献

[1] W. Saric, H. L. Reed and E. B. White, “Stability and transition of three-dimensional boundary layers,” Annu. Rev. Fluid Mech. 35, 413–440 (2003).
[2] R.S. Downs and E.B. White, “Free-stream turbulence and the development of cross-flow disturbances,” J. Fluid Mech. 735, 347–380 (2013).
[3] V.I. Borodulin, A.V. Ivanov, Y.S. Kachanov and A. Hanifi, “Laminar-turbulent transition delay on a swept wing,” AIP Conf. Proc. 1770(1), 030065 (2016).
[4] J. Sepieri and M. Kotosonis, “Three-dimensional organisation of primary and secondary crossflow instability,” J. Fluid Mech. 799, 200–245 (2016).
[5] L. De Vincentiis, D. Henningson and A. Hanifi, “Transition in an infinite swept-wing boundary layer subject to surface roughness and free-stream turbulence,” J. Fluid Mech. 931, A24 (2022).
[6] A. Yakeno and S. Obayashi, “Propagation of stationary and traveling waves in a leading-edge boundary layer of a swept wing,” Phys. Fluids 33 (9), 094111 (2021).
[7] M. Högberg and D.S. Henningson, “Secondary instability of cross-flow vortices in Falkner–Skan–Cooke boundary layers,” J. Fluid Mech. 368, 339–357 (1998).
[8] M. Brynjell–Rahkola, N. Shahriari, P. Schlatter, A. Hanifi and D.S. Henningson, “Stability and sensitivity of a cross-flow-dominated Falkner–Skan–Cooke boundary layer with discrete surface roughness,” J. Fluid Mech. 826, 830–850 (2017).
[9] T. Ishida, T. Tsukahara and N. Tokugawa, “Parameter effects of spanwise-arrayed cylindrical roughness elements on transition in the Falkner–Skan–Cooke boundary layer,” Trans. Japan Soc. Aeronautical and Space Sciences, 65(2), 84–94 (2022).
[10] K. Nakagawa, T. Tsukahara and T. Ishida, “DNS study on turbulent transition induced by an interaction between freestream turbulence and cylindrical roughness in swept flat-plate boundary layer,” Aerospace, 10, 128 (2023).
[11] H. Kurz and M. Kloker, “Mechanisms of flow tripping by discrete roughness elements in a swept-wing boundary layer,” J. Fluid Mech. 796, 158–194 (2016).
[12] D. Watanabe and H. Maekawa, “Rapid growth of unsteady finite-amplitude perturbations in a supersonic boundary-layer flow,” Proc. 9th International Symposium on Turbulence, Shear Flow Phenomena (2015).

[13] P. Wassurmann and M. Kloker, “Mechanisms and passive control of crossflow-vortex-induced transition in a three-dimensional boundary layer,” J. Fluid Mech. 456, 49–84 (2002).
[14] P. Wassurmann and M. Kloker, “Transition mechanisms induced by travelling crossflow vortices in a three-dimensional boundary layer,” J. Fluid Mech. 483, 67–89 (2003).
[15] J. Casacuberta, K.J. Groot, S. Hickel and M. Kotsonis, “Secondary instabilities in swept-wing boundary layers: Direct numerical simulations and biglobal stability analysis,” AIAA SciTech 2022 Forum p.2330 (2022).
[16] S.K. Robinson, “Coherent motions in the turbulent boundary layer,” Annu. Rev. Fluid Mech. 23, 601–639 (1991).
[17] A. Vaid, N.R. Vadlamani, A.S. Malathi and V. Gupta, “Dynamics of bypass transition behind roughness element subjected to pulses of free-stream turbulence,” Phys. Fluids 34, 114110 (2022).

［報 告］

オープンキャンパス 2023 報告

情報セキュリティ研究部 水木敬明

2023 年 7 月 26 日と 27 日の 2 日間，東北大学オープンキャンパス 2023 が開催されました．来場制限を行わない通常開催のオープンキャンパスとしては 4 年ぶりとなり，2 日間で約 58,000 人が本学に来場されたとのことです．

サイバーサイエンスセンターにおきましても 4 年ぶりに対面での一般公開となりました．具体的には次のような要領で当センターのオープンキャンパスを実施しました．

> 東北大学サイバーサイエンスセンターでは，7 月 26 日，27 日両日にかけてオープンキャンパスを開催致します．スーパーコンピュータ AOBA や研究部の最新の研究成果に関する展示をご覧いただけます．東北大学オープンキャンパスにお越しになる際は、是非、サイバーサイエンスセンターにお立ち寄りください．
>
> 【開催日時】2023 年 7 月 26 日（水）10:30〜14:30
> 2023 年 7 月 27 日（木）10:30〜14:30
> 【内容】　スーパーコンピュータ AOBA の見学
> 研究部による研究紹介

2019 年度以前より開催時間や場所に関して規模を縮小しつつ，本センターにご興味を持つ方々にじっくりご見学いただくことができたと考えております．ご来場いただきました皆様に感謝申し上げるとともに，来年度の参加をお待ちしております．

［報 告］

新スーパーコンピュータ「AOBA-1.5」導入披露式典開催報告

高橋 慧智
東北大学サイバーサイエンスセンター スーパーコンピューティング研究部

2023 年 9 月 22 日，東北大学サイバーサイエンスセンター （以下本センター） において新スーパーコンピュータ「AOBA-1.5」導入披露式典を開催しました．本センターでは 2023 年 8 月より新サブシステム「AOBA-S」の運用を開始し，既設サブシステムである AOBA-A/B と合わせた本センターのスーパコンピュータを AOBA-1.2 から AOBA-1.5 へとアップデートしました．本式典は，AOBA-1.5 の本格運用開始を記念して開催されたものです．

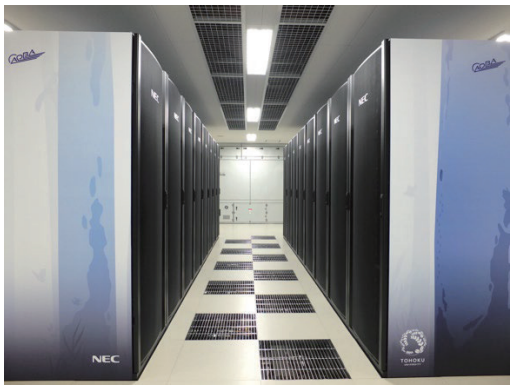


図 1 新サブシステム「AOBA-S」

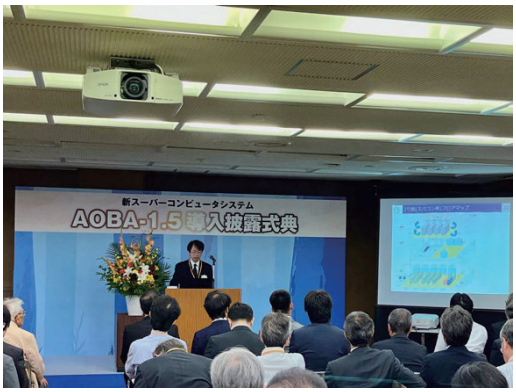


図 2 導入披露式典の様子

AOBA-S は 504 台の NEC 製最新ベクトル型スーパコンピュータシステム「SX-Aurora TSUBASA C401-8」から構成され，計 21.05 PFLOP/s の演算性能と 9.97 PB/s のメモリ帯域幅を備えています．AOBA-A と比較すると演算性能では約 14 倍，メモリ帯域幅では約 11 倍の性能向上となり，従来より格段に大規模かつ高精細な数値シミュレーションが可能になります．また，AOBA-S は本学青葉山新キャンパスで整備が進められている次世代放射光施設「ナノテラス」と高速ネットワークによって直結され，実験機器から生成される膨大な観測データを蓄積・解析するための受け皿としての機能も期待されます．

導入披露式典は本センター本館 5 階大会議室において開催され，文部科学省研究振興局参事官（情報担当）をはじめ，本学理事，NEC の来賓，各国立大学や国立研究所の情報基盤センターの関係者を始めとし，約 80 名の方々にご出席いただきました．まず本センター長 菅沼拓夫教授がセンターの教職員を代表して式辞を述べた後，本学理事・副学長 青木孝文教授より挨拶がありました．その後，文部科学省研究振興局参事官（情報担当）嶋崎政一氏，NEC Corporate EVP 雨宮邦和氏，東京大学情報基盤センター長 田浦健次朗教授からそれぞれ祝辞をいただきました．最後に本センター副センター長 滝沢寛之教授より，AOBA-1.5 導入の経緯とシステムの概要について説明がありました．式典終了後にはスーパーコンピュータが設置されている本センター2 号館 1 階において新スーパーコンピュータの見学会が行われました．

見学会に続いて，再び本館大会議室において記念講演会が行われ，3 名の講演者にご登壇いただきました．名古屋工業大学大学院 平田晃正教授は，本センター教員との共同研究でもある熱中症リスク評価シミュレーションや，新型コロナウィルス感染者数の予測に関する研究に関してご講演い

ただきました．海洋研究開発機構 付加価値情報創生部門 地球情報科学技術センター長 石川洋一氏には，気候変動シミュレーションにおける高性能計算の重要性と，AOBA-S の計算能力に対する期待に関してご講演いただきました．NEC Corporate SVP 兼インフラ・テクノロジーサービス事業部門長 木村哲彦氏には，SENAC-1 の共同開発から 60 年以上に渡り続く NEC と本センターの高性能計算技術に関する産学連携の取り組みに加え，量子計算を始めとする未来のコンピューティング技術に関して展望を述べられました．

講演会終了後には引き続き，青葉山コモンズみどり食堂において交流会が開かれ，工学部・工学研究科長 伊藤彰則教授のご発声で乾杯を行いました．終始和やかな雰囲気の中で，各大学の情報基盤センター長や NEC のご来賓など学内外の方々からスピーチをいただき，盛況のうちに閉会しました．

［報 告］

後藤英昭准教授がWireless Broadband Alliance(WBA)にて Contributor Award 2022を受賞

当センターの後藤英昭准教授（ネットワーク研究部）がWireless Broadband Alliance(WBA)にてContributorAward 2022を受賞しました。
WBAにおける無線LANローミングシステムの技術・仕様の開発や、国内外のOpenRoaming普及などの貢献が、高く評価されたものです。

【WBA RECOGNITION AWARDS】
https://wballiance.com/recognition-awards/

クラウドサービス基盤研究室（CSI研究室）がWireless Broadband Alliance(WBA)にて紹介されました

クラウドサービス基盤研究室 (CSI 研究室)の研究および社会実装の活動について、Wireless Broadband Alliance(WBA)より紹介記事が発行されました。

・WBA Case Studies: Driving Connectivity in Smart Cities: Cityroam’s OpenRoaming™ Revolution
https://wballiance.com/resource/driving-connectivity-in-smart-cities-cityroams-openroaming-revolution/

・WBA ‘Meet the Experts’: Dr. Hideaki Goto
https://wballiance.com/meet-the-wba-experts-dr-hideaki-goto/

［スーパーコンピュータ AOBA のお知らせより］

東北大学サイバーサイエンスセンター大規模科学計算システムウェブサイトに掲載されたお知らせの一部を転載しています。
https://www.ss.cc.tohoku.ac.jp/information/

コンパイラのバージョンアップについて

2023 年 9 月 4 日に AOBA のコンパイラをバージョンアップいたします。

システム	コンパイラ名	旧バージョン	新バージョン	ドキュメント
AOBA-S	Fortran Compiler	5.0.1	5.0.2	マニュアル＆リリースノート
	C/C++ Compiler	5.0.1	5.0.2	
AOBA-A	Fortran Compiler	4.0	5.0.2	
	C/C++ Compiler	4.0	5.0.2	
	MPI※1	3.2.0	3.4.0	
AOBA-B	Intel Compiler ※2、※3	oneAPI 2023.0	oneAPI 2023.2	oneAPI マニュアル関連

※1 MPI を利用するプログラムは再コンパイルが必要
※2 Intel oneAPI 2023.2 の環境変数設定ファイルは、bash 向けのみの提供
※3 HPCI 用フロントエンドサーバの OS が oneAPI 2023.2 に未対応のため、AOBA のフロントエンドサーバをご利用ください。

（共同利用支援係, 共同研究支援係）

商用アプリケーションのバージョンアップについて

数式処理システム「Mathematica」のバージョンアップを行いましたのでお知らせいたします。新機能の概要、機能の詳細については開発元 Web サイトをご参照ください。

Mathematica

- バージョン：13.3
- バージョンアップ日：2023 年 9 月 4 日
- サービスホスト：フロントエンドサーバ
- 起動コマンド：(GUI 版) mathematica　(コマンドライン版) math
- 開発元 Web サイト：https://www.wolfram.com/mathematica/new-in-13/

（共同利用支援係）

学部学生のためのスーパーコンピュータ無償提供制度について

東北大学サイバーサイエンスセンターでは、学部学生(3 年生、4 年生)が、卒業論文等作成のために大規模科学計算システムを無料利用できる制度を実施いたします。希望者は以下を確認頂き申請書に必要事項を記入の上、お申し込みください。

本センター教員が内容を審査の上、採択となった研究課題については、以下の期間大規模科学計算システムを無料で利用する(利用ノード時間に上限あり)ことができます。

1. 応募期間

~~・第一回　令和 5 年 10 月 1 日(日)～令和 5 年 10 月 21 日(土)~~

・第二回　令和 5 年 12 月 1 日(金)～令和 5 年 12 月 21 日(木)

2. 利用期間

採択日～令和 6 年 3 月下旬

3. 応募詳細

・研究成果を学術論文誌等において発表する場合は、謝辞等で本センターの貢献を明記してください。

・年度末に成果報告書を提出して頂きます。

・申し込みには指導教員の承認が必要となります。

・高等専門学校生については本科 5 年生および専攻科生を対象といたします。

・指導教員 1 人につき最大 2 件までの応募となります。

4. 応募方法

応募される方は、本センターのウェブサイト (https://www.ss.cc.tohoku.ac.jp/) の「各種申請用紙」から「学部学生のためのスーパーコンピュータ無償提供制度申請書」をダウンロードし、必要事項を記入して電子メールでお申し込みください。

(送り先)E-mail:edu-prog@cc.tohoku.ac.jp

5. 問い合わせ先

共同利用支援係

TEL :(022)795-6251

E-mail :cc-uketuke@grp.tohoku.ac.jp

(共同利用支援係)

— SENAC 執筆要項 —

１．お寄せいただきたい投稿内容

サイバーサイエンスセンターでは、研究者・技術者・学生等の方々からの原稿を募集しております。以下の内容で募集しておりますので、皆さまのご投稿をお待ちしております。なお、一般投稿いただいた方には、謝礼として負担金の一部を免除いたします。

・一般利用者の方々が関心をもたれる事項に関する論説
・センターの計算機を利用して行った研究論文の概要
・プログラミングの実例と解説
・センターに対する意見、要望
・利用者相互の情報交換

２．執筆にあたってご注意いただく事項

(1)原稿は横書きです。
(2)術語以外は、「常用漢字」を用い、かなは「現代かなづかい」を用いるものとします。
(3)学術あるいは技術に関する原稿の場合、200 字～400 字程度のアブストラクトをつけてください。
(4)参考文献は通し番号を付し末尾に一括記載し、本文中の該当箇所に引用番号を記入ください。
　・雑誌：著者, タイトル, 雑誌名, 巻, 号, ページ, 発行年
　・書籍：著者, 書名, ページ, 発行所, 発行年

３．原稿の提出方法

原稿のファイル形式は Word を標準としますが、PDF での提出も可能です。サイズ*は以下を参照してください。ファイルは電子メールで提出してください。

－用紙サイズ・文字サイズ等の目安－

・サイズ：A4
・余白：上＝30㎜　下＝25㎜　左右＝25㎜　綴じ代＝0
・標準の文字数（45 文字 47 行）
・表題＝ゴシック体 14pt 中央　・副題＝明朝体 12pt 中央
・氏名＝明朝体 10.5pt 中央
・所属＝明朝体 10.5pt 中央
・本文＝明朝体 10.5pt
・章・見出し番号＝ゴシック体 11pt～12pt
　*余白サイズ、文字数、文字サイズは目安とお考えください。

４．その他

(1)一般投稿を頂いた方には謝礼として、負担金の一部を免除いたします。免除額は概ね 1 ページ 1 万円を目安とします。詳細は共同利用支援係までお問い合わせください。
(2)投稿予定の原稿が 15 ページを超す場合は共同利用支援係まで前もってご連絡ください。
(3)初回の校正は、執筆者が行って、誤植の防止をはかるものとします。
(4)原稿の提出先は次のとおりです。

東北大学サイバーサイエンスセンター内　情報部デジタルサービス支援課共同利用支援係
e-mail　cc-uketuke@grp.tohoku.ac.jp
TEL　　022-795-3406

スタッフ便り

歳を重ねるごとに、おなか周りが分厚くなってきました。若いころは、内臓脂肪の CM をみても、自分には無関係なのではと流していましたが、やはり誰もが通る道なのかと実感しています。一時期、このままではまずいと思い、自宅で腕立て・腹筋をやる時期もあったのですが、今では毎日夜 10 時になると「自宅トレーニングの時間です」とアラームがなるものの、完全にスルーしている自分がいます。「継続は力なり」とは言ったもので、数日、やらない日が続いてしまうとそこから先は、もうやらなくてもいいモードになってしまいます。これを打破するには、大きな変化が必要です。期待しているのは自分の子供たちが何かスポーツを始めてくれて、一緒に体を動かすことです。最近水泳教室に通い始めましたが、水泳だと一緒にできないので、そのうち球技にも興味をもってもらいたいなと思っています。完全に他人頼みになっていますが、自分の健康は自分で何とかしていきたいものです。(Y. S)

スーパーコンピュータ「AOBA-1. 5」として新システムの運用が始まりました。新たに導入された AOBA-S には、動物や植物が隠れているのをご存知でしょうか。もちろん本物ではなくサーバラックにデザインされた動植物たちです。青葉山をモチーフとした森の中に、何種類かの動植物が見え隠れしています。リモートからの利用では体験できない「生(なま) AOBA」も現地で探究してみてはいかがでしょうか。(K. O)

【サイバーサイエンスセンタースタッフ採用のお知らせ】

2023. 10. 1　採用

宗形　聡 特任准教授(データプラットフォーム研究部)

SENAC 編集部会

滝沢寛之　水木敬明　後藤英昭　高橋慧智
今野義則　早坂和勝　大泉健治　小野　敏
斉藤くみ子

令和 5 年 10 月発行
編集・発行　東北大学
サイバーサイエンスセンター
仙台市青葉区荒巻字青葉 6-3
郵便番号　980-8578
PDF 作成　株式会社　東誠社

スーパーコンピュータ AOBA システム一覧

計算機システム	機　種
サブシステム AOBA-S	SX-Aurora TSUBASA Type 30A
サブシステム AOBA-A	SX-Aurora TSUBASA Type 20B
サブシステム AOBA-B	LX 406Rz-2

サーバとホスト名

フロントエンドサーバ（AOBA-S 用）	sfront.cc.tohoku.ac.jp
データ転送サーバ（AOBA-S 用）	sfile.cc.tohoku.ac.jp
ログインサーバ（AOBA-A,B 用）	login.cc.tohoku.ac.jp
データ転送サーバ（AOBA-A,B 用）	file.cc.tohoku.ac.jp

サービス時間

利用システム名等	利用時間帯
サブシステム AOBA-S	連　続　運　転
サブシステム AOBA-A	連　続　運　転
サブシステム AOBA-B	連　続　運　転
各種サ ー バ	連　続　運　転
大判プリンタ	平日 9:00～21:00

クラウドサービス AOBA-S の利用形態と制限値

利用形態	キュー名	VE 数※	実行形態	最大経過時間 既定値/最大値	メモリサイズ
無料	sxsf	1	1VE	1 時間/1 時間	96GB
共有	sxs	1	1VE	72 時間/720 時間	96GB×VE 数
		1～2048	8VE 単位で確保		
占有	個別設定				

※2VE以上を利用した並列実行にはMPIの利用が必用

サブシステム AOBA-A の利用形態と制限値

利用形態	キュー名	VE 数※	実行形態	最大経過時間 既定値/最大値	メモリサイズ
無料	sxf	1	1VE	1 時間/1 時間	48GB
共有	sx	1	1VE	72 時間/720 時間	48GB×VE 数
		2～256	8VE 単位で確保		
占有	個別設定				

※2VE以上を利用した並列実行にはMPIの利用が必用

サブシステム AOBA-B の利用形態と制限値

利用形態	キュー名	ノード数※	最大経過時間 既定値/最大値	メモリサイズ
共有	lx	1～16	72 時間/720 時間	256GB×ノード数
占有	個別設定			

※2ノード以上を利用した並列実行にはMPIの利用が必用

目次

東北大学サイバーサイエンスセンター
大規模科学計算システム広報 Vol.56 No.4 2023−10

［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　1
SX-Aurora TSUBASA でのプログラミング（並列化編）…… 林　康晴　17
早坂　武
サブシステムAOBA-S の利用法 …… 40

［共同研究成果］
後退平板境界層における横流れ渦と主流乱れの相互作用による乱流遷移の直接数値解析 …… 中川　皓介　64
塚原　隆裕

［報 告］
オープンキャンパス2023報告 …… 水木　敬明　74
新スーパーコンピュータ「AOBA-1.5」導入披露式典開催報告 … 高橋　慧智　75
後藤英昭准教授がWireless Broadband Alliance(WBA)にてContributor Award 2022を受賞 …… 77
クラウドサービス基盤研究室（CSI研究室）がWireless Broadband Alliance(WBA)にて紹介されました …… 77

［スーパーコンピュータAOBAのお知らせより］
コンパイラのバージョンアップについて …… 78
商用アプリケーションのバージョンアップについて …… 78
学部学生のためのスーパーコンピュータ無償提供制度について …… 79

執筆要項 …… 80

スタッフ便り …… 81