2021 年度
SX-Aurora TSUBASAにおける
性能分析と高速化

2021 年　9 月 28 日

東北大学サイバーサイエンスセンター
日本電気株式会社

本資料は，東北大学サイバーサイエンスセンターとNECの共同により作成されたものです.

目　次

演習問題の構成

演習問題の環境を自分のホームディレクトリ配下にコピーします。

/mnt/stfs/ap/lecture/TUNE/

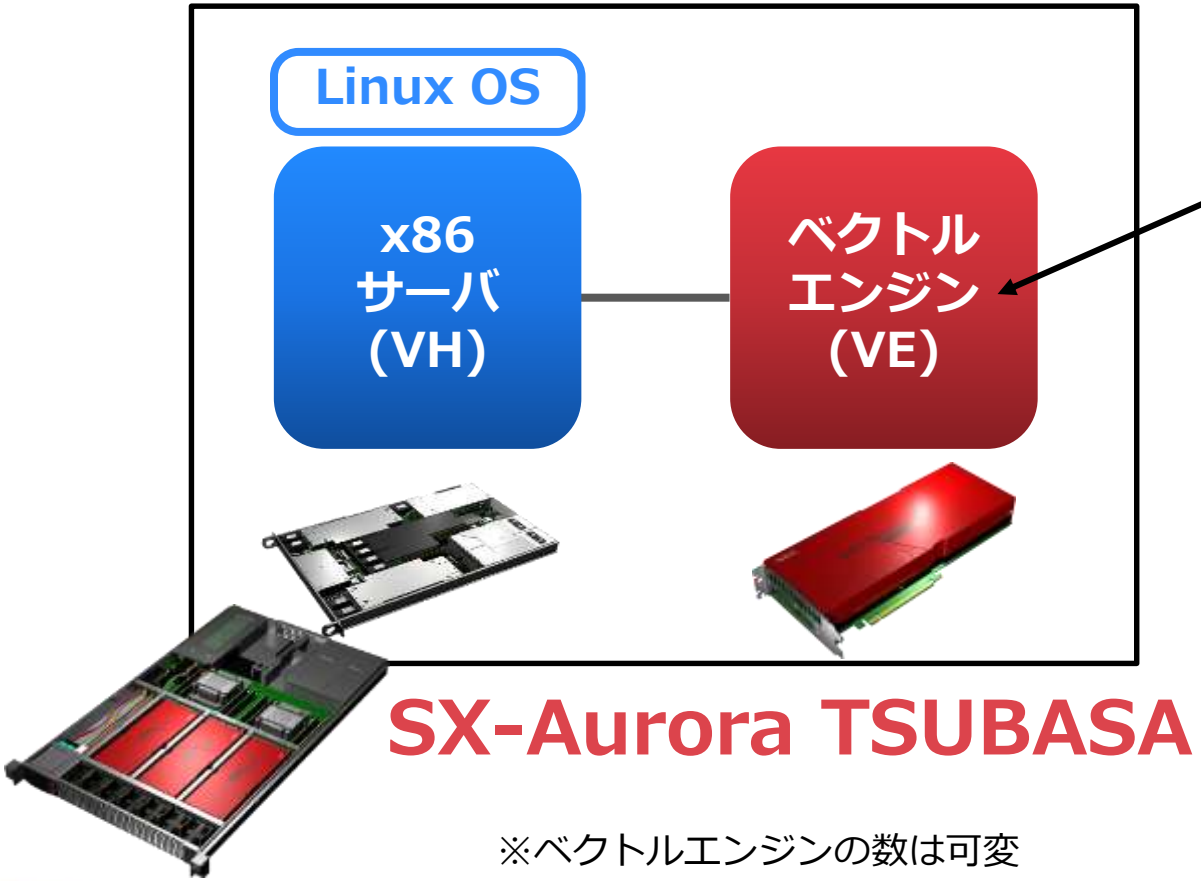
|-- practice_1　演習問題1
|-- practice_2　演習問題2
|-- practice_3　演習問題3

```
$ cd <環境をコピーしたいディレクトリ>
$ cp –r /mnt/stfs/ap/lecture/TUNE/ .
```

1． SX-Aurora TSUBASA概要

SX-Aurora TSUBASAアーキテクチャ

SXシリーズの特徴であるベクトルプロセッサをPCIeカード型のベクトルエンジンとして搭載



ベクトルプロセッサ

※ベクトルエンジンの数は可変

ベクトルエンジンの特徴

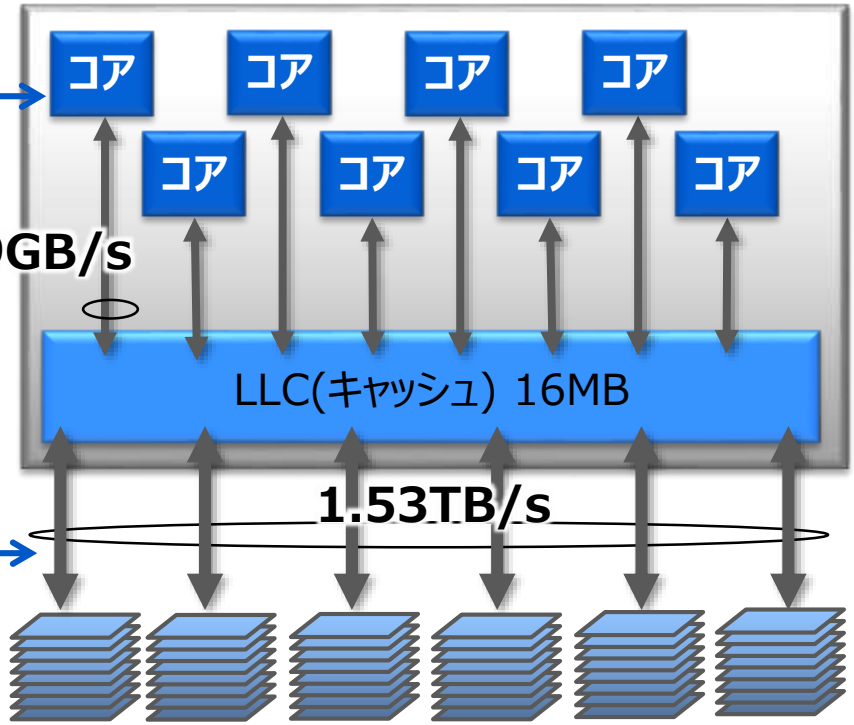
プロセッサ

- **世界最速クラスのコア**
  - 307GFlops (DP)
  - 614GFlops (SP)
- **世界最速クラスのデータアクセス性能**
  - 1.53TB/s
- **テクノロジ**
  - 世界初HBM2 x6実装

ビックコア

No.1

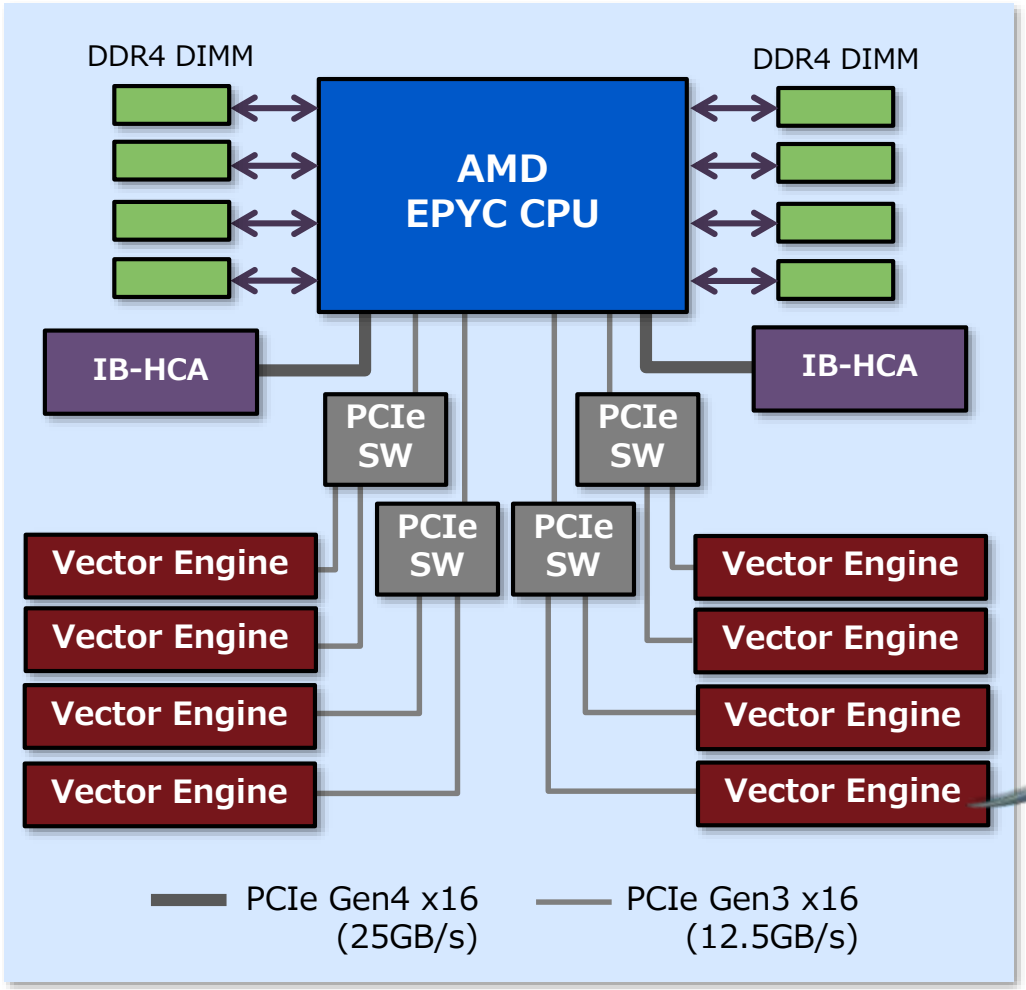
世界初となるCPUと6個の3次元積層メモリHBM2搭載技術をTSMC社と共同開発



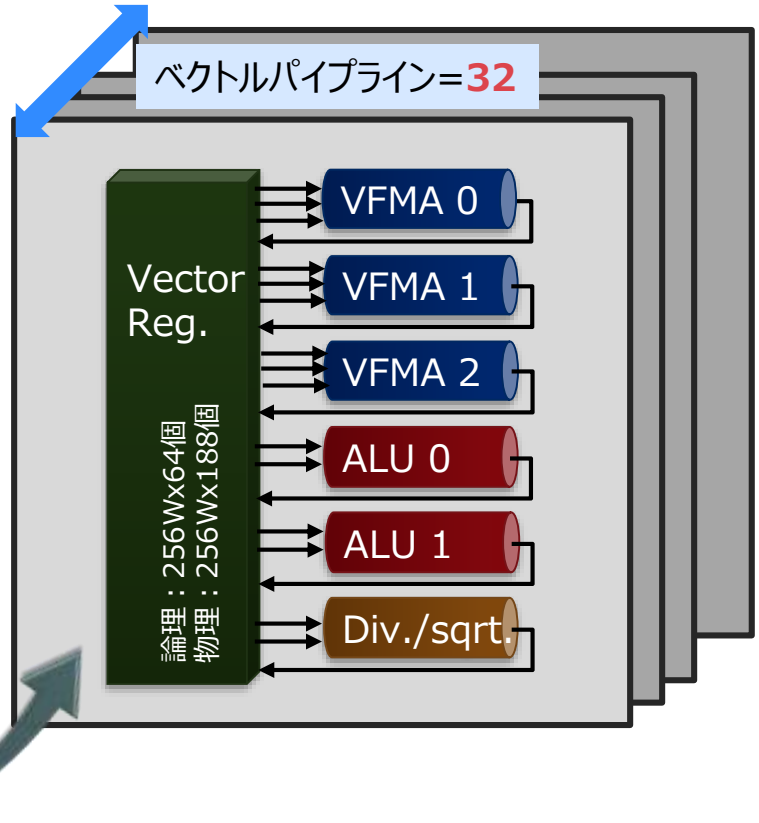
性能諸元	SX-AuroraTSUBASA
Core性能[GF]	307.2
Core数/CPU	8
CPU性能[TF]	2.45
メモリ BW[TB/s]	1.53
メモリ容量[GB]	48

SX-Aurora TSUBASAの構成

SX-Aurora TSUBASA ブロック図



SX-Aurora TSUBASA ベクトル処理部



SX-Aurora TSUBASAのベクトル処理部

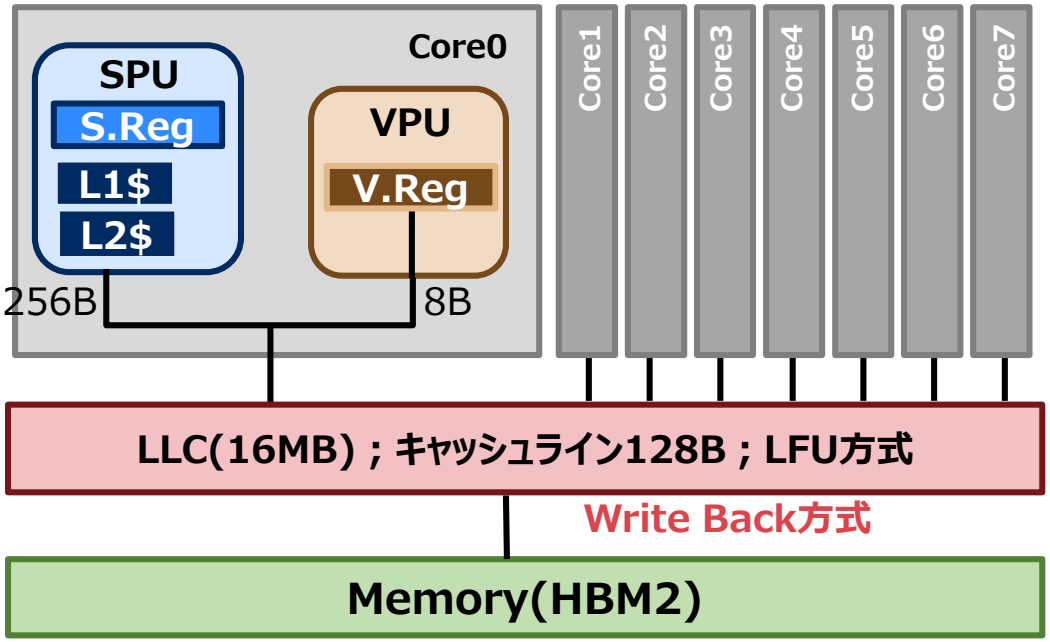
- **ベクトルレジスタ**
  - 物理レジスタを設けることにより、レジスタリネーミング方式を採用し、命令追い越しを強化
- **FMA演算器を採用**
  - 丸め誤差が気になる場合はコンパイラオプションでFMA演算を抑止することが可能 (**-mno-vector-fma**)
- **逆数近似処理のサポート**
  - 除算/SQRT演算器をHW実装しているが、更に高速に処理するために逆数近似処理をサポート
    - 除算：逆数近似でも誤差なし
    - SQRT:演算結果の誤差あり
  - コンパイラオプションにより高速処理版の逆数近似処理をサポート(**-mvector-low-precise-divide-function**)
    - 除算結果の仮数部に最大１ビットの誤差が含まれる場合がある
  - 除算/SQRT演算器を使用する場合はコンパイラオプションで指定可能
    - SQRTの計算誤差が問題になる場合
    - FMA演算器を他の処理で使用した方が性能が上がる場合

      ベクトル浮動小数点除算において、除算演算器を使用する場合：**-mvector-floating-divide-instruction**

      SQRT処理を除算演算器を使用する場合：**-mvector-sqrt-instruction**
- **ベクトル総和演算の計算結果について**
  - 今までのSXシリーズと同様、ベクトル総和演算の計算結果に差分が生じる可能性あり（加算順序の違いが原因）
  - ベクトル総和演算を使用しない場合はオプションで指定可能(**-mno-vector-reduction**)
    - 本オプションを使用するとベクトル化不可となるので、総和処理のコストが重い場合は実行時間が増大するため注意が必要

SX-Aurora TSUBASAのメモリ階層

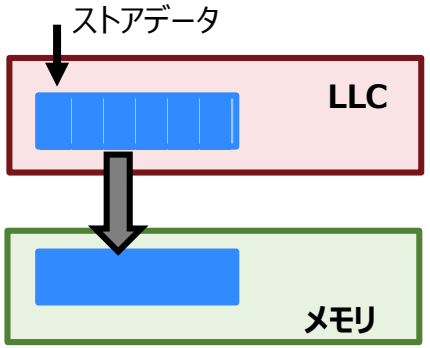
L1$I :32KB
L1$O :32KB
L2$:256KB



・データ置き換えアルゴリズムは
LFU(Least Frequently Used)
・コンパイル時に制御ビットを付与し、
その制御ビットによりアクセス頻度を
調整することが可能
・制御ビットはユーザが制御可能

SX-Aurora TSUBASAのLLCのWrite Back方式について

- LLC容量があふれた場合(キャッシュインデックスが競合した場合)にストアデータをLLC→メモリに書き出す
- 128Bのキャッシュラインのデータがすべて有効の場合で、かつメモリアクセスが行われていないタイミングでストアデータをメモリに書き出す
- LLCからメモリに書き出す際に、128Bのキャッシュラインがすべて有効データか否かをHWが自動的に判断し、128Bすべてが有効データの場合はそのままメモリに書き出し、無効データが含まれる場合は対象ブロックデータをメモリからロードし無効データを有効データに書き換えてから128Bをメモリに書き出す



Write Back方式の効果

(短VL+連続アクセス)性能 vs. (長VL+ストライドアクセス)性能

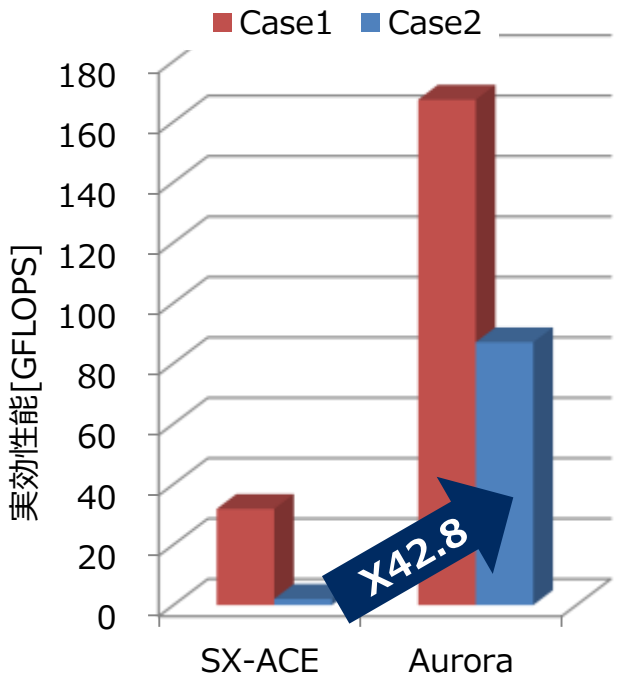
●例：

Case1:
短VL＋連続アクセス

```
do 10 k=0,Nz
  do 10 j=0,Ny
    do 10 i=0,Nx (Nx=100)
      E_x(i,j,k)=
     &       C_x_a(i,j,k)*E_x(i,j,k)
     &      +C_x_b(i,j,k)*( ( H_z(i,j,k)-H_z(i,j-1,k  ) )/dy
     &                    -( H_y(i,j,k)-H_y(i,j  ,k-1) )/dz
     &                    -E_x_Current(i,j,k)
     &                   )
  :
  :
10 CONTINU
```

Case2:
長VL＋ストライドアクセス

```
do 10 k=0,Nz
  do 10 i=0,Nx
    do 10 j=0,Ny (Ny=750)
      E_x(i,j,k)=
     &       C_x_a(i,j,k)*E_x(i,j,k)
     &      +C_x_b(i,j,k)*( ( H_z(i,j,k)-H_z(i,j-1,k  ) )/dy
     &                    -( H_y(i,j,k)-H_y(i,j  ,k-1) )/dz
     &                    -E_x_Current(i,j,k)
     &                   )
  :
  :
10 CONTINU
```

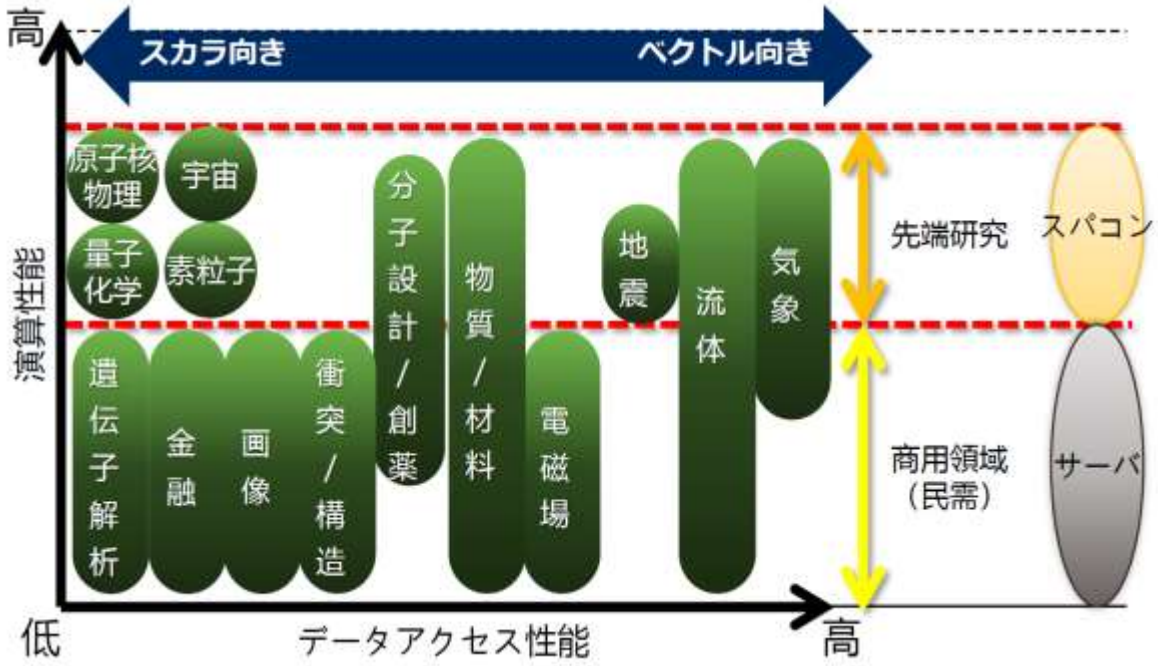


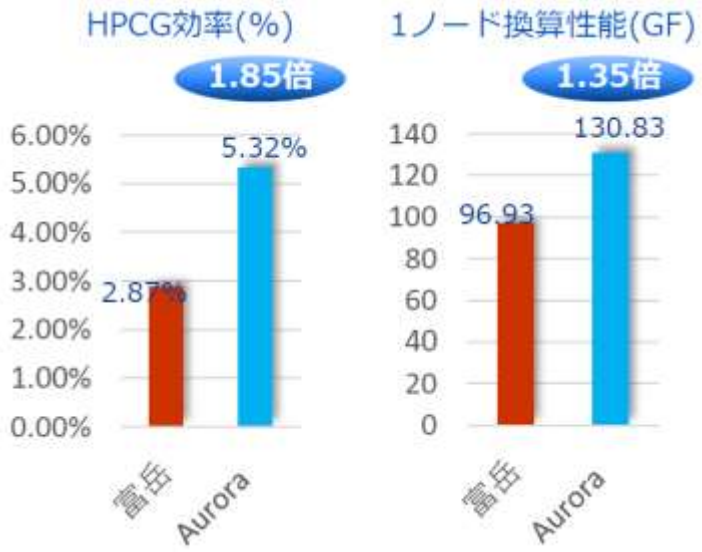
Case1とCase2の性能差は**1.9倍**
→Write Backにより性能低下を抑制
→Case1よりCase2の性能が低いのはストライドアクセスによりLLCが乱されるため

アプリケーション特性と計算機適正

ベクトル型スーパーコンピュータSX-Aurora TSUBASAが適する領域

- 高メモリバンド幅を有するSX-Aurora TSUBASAは，メモリからのデータ供給の要求が大きいシミュレーション領域で高い性能を発揮





- HPCG実効性能
  - ✓ HPCGは有限要素法から得られる疎行列を対象とした線形ソルバーを評価する実アプリケーションに近いベンチマークプログラム。
  - ✓ SX-Aurora TSUBASAは富岳より高い効率でHPCGを実行

2． SX-Aurora TSUBASA性能分析ツール

利用可能な性能分析ツール

高速化を実施する上で利用可能な性能分析ツールは以下

- コンパイルメッセージ(ベクトル化および並列化診断メッセージ)
- コンパイルリスト(編集リスト)
- プログラム情報
- プロファイラ
- 簡易性能解析(ftrace,ftraceviewer)

Fortranコンパイラの利用

```
$ nfort -mparallel -O3 a.f90 b.f90
```
… Fortranプログラム(a.f90, b.f90)のコンパイル、リンク

-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
-O1 … 副作用のない自動ベクトル化を適用
-O0 … ベクトル化、最適化を行わない

高
低

これらは、コンパイラの自動ベクトル化、最適化レベルをコントロールする、

-fopenmp … OpenMP Fortran機能を利用
-mparallel … 自動並列化機能を利用

これらは、コンパイラの並列処理機能をコントロールする。
並列処理機能を使用しないときは指定しなくてよい。

代表的なコンパイラオプションの指定例

`$ nfort a.f90`	既定レベルの自動ベクトル化を適用し、コンパイル、リンク
`$ nfort -O4 a.f90 b.f90`	最大レベルの自動ベクトル化を適用し、複数のプログラムをコンパイル、リンク
`$ nfort -mparallel -O3 a.f90`	自動並列化、および、高度なレベルの自動ベクトル化を適用し、コンパイル、リンク
`$ nfort -O4 -finline-functions a.f90`	自動インライン展開、および、最大レベルの自動ベクトル化を適用し、コンパイル、リンク
`$ nfort -O0 -g a.f90`	ベクトル化を止めて、シンボリックデバッグするコンパイル、リンク
`$ nfort -g a.f90`	ベクトル化を止めずに、シンボリックデバッグするコンパイル、リンク
`$ nfort -E a.f90`	プリプロセスのみ実行。プリプロセス結果は標準出力に出力する
`$ nfort -fsyntax-only a.f90`	シンタックスチェックのみ実行

ベクトル化(並列化)診断メッセージ

- コンパイラの出力するメッセージ、リストにより、ループのベクトル化状況、ベクトル化不可原因を調べることができる
  - 標準エラー出力　　… **-fdiag-vector=2** (詳細情報出力)
  - リストファイル出力　　… **-report-diagnostics**

```
$ nfort –fdiag-vector=2 abc.f
…
nfort: vec( 103): abc.f, line 23: Unvectorized loop.
nfort: vec( 122): abc.f, line 24: Dependency unknown. Unvectorizable dependency is assumed.: RHO
nfort: vec( 122): abc.f, line 25: Dependency unknown. Unvectorizable dependency is assumed.: RHO
nfort: vec( 101): abc.f, line 50: Vectorized loop.
…
$ nfort –report-diagnostics abc.f
…
$ less abc.L
FILE NAME: abc.f
…
PROCEDURE NAME: SUB
DIAGNOSTIC LIST

 LINE              DIAGNOSTIC MESSAGE

    23: vec( 103): Unvectorized loop.
    24: vec( 122): Dependency unknown. Unvectorizable dependency is assumed.: RHO
    25: vec( 122): Dependency unknown. Unvectorizable dependency is assumed.: RHO
    50: vec( 101): Vectorized loop.
…
```

ベクトル化不可と思われる依存関係が変数RHOにあったとみなし、ベクトル化しなかったことを示すメッセージ

リストファイル名は「ソースファイル名.L」

編集リスト (1/5)

ソース行とともにループ構造、そのベクトル化状況などを記号で表示

- **-report-format**または**-report-all**が指定されたとき出力

```
$ nfort -report-format a.f90 -c
...
$ less a.L
          :
PROCEDURE NAME: SUB
FORMAT LIST

 LINE   LOOP      STATEMENT

     1:           SUBROUTINE SUB(A, B, C, X, Y, Z, N)
     2:             INTEGER :: N
     3:             REAL(KIND=4) :: A(N), B(N), C(N)
     4:             REAL(KIND=16) :: X(N), Y(N), Z(N)
     5:             INTEGER :: I
     6:
     7: V------>     DO I = 1, N
     8: |             A(I) = B(I) * C(I)
     9: V------      END DO
    10:
    11: +------>     DO I = 1, N
    12: |             X(I) = Y(I) * Z(I)
    13: +------      END DO
    14:
    15:           END SUBROUTINE SUB
```

リストファイル名は「ソースファイル名.L」

ベクトル化されたループ

ベクトル化されなかったループ

編集リスト (2/5)

◆ ループ全体がベクトル化された場合

```
V------> DO I=1,N
|
V------  END DO
```

◆ ベクトル化されなかった場合

```
+------> DO I=1,N
|
+------  END DO
```

◆ ループが部分ベクトル化された場合

```
S------> DO I=1,N
|
S------  END DO
```

◆ ループが条件ベクトル化された場合

```
C------> DO I=1,N
|
C------  END DO
```

Auroraで追加

◆ ループが並列化された場合

```
P------> DO I=1,N
|
P------  END DO
```

◆ ループが並列化、かつベクトル化された場合

```
Y------> DO I=1,N
|
Y------  END DO
```

編集リスト (3/5)

◆ 配列式など、一行にループ全体が含まれるとき

```
V======> A = A + B
```

ループの先頭と最後の行が同じである場合，
ループの構造は “=” で表示される

◆ 手続呼出しがインライン展開された場合

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

インライン展開された手続がある行には “I” が表示される

◆ 多重ループが一重化された場合

```
W------>   do J =1,N
|*---->     do I =1,M
||
|*----      enddo
W------    enddo
```

一重化されたループの外側ループに “W”， 内側ループに“＊”が表示される

編集リスト (4/5)

◆ループが入れ換えられ、ベクトル化がされた場合

```
X------>    do J =1,1000
|*----->      do I =1,10
||
|*-----       enddo
X------     enddo
```

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

◆ループが融合された場合

```
V------> DO I=1,N
|
|         END DO
|         DO I=1,N
|
V------   END DO
```

編集リスト (5/5)

◆ 外側ループがループアンローリングされ、内側ループがベクトル化された場合

```
U------> DO I=1,N                    Auroraで追加
|V------>    DO J=1
|
|
|V-----      END DO
U------  END D
```

◆ ループが展開された場合

```
*------> DO I=1,4
|
*------  END DO
```

- 17カラム目に表示される文字は行がどう最適化されたかを表す
  - “M” この行を含む多重ループがベクトル行列積ライブラリ呼び出しに置き換えられた
  - “F” 式に対してベクトル積和命令が生成された Auroraで追加
  - “R” 配列にretain指示行が適用された Auroraで追加
  - “G” ベクトル収集命令が生成された
  - “C” ベクトル拡散命令が生成され
  - “V” 配列にvreg指示行、または、pvreg指示行が適用された Auroraで追加

プログラム情報 (1/4)

プログラム全体の性能情報を採取

- 実行時に環境変数**VE_PROGINF**に以下のどちらかの値をセット

  “**YES**” … 基本情報

  “**DETAIL**” … 基本情報+メモリ情報(※1) (※2はDETAILかつマルチスレッド実行時)

```
 ********  Program  Information  ********
Real Time (sec)                          :           33.060014
User Time (sec)                          :          109.886525
Vector Time (sec)                        :          104.243699
Inst. Count                              :        77145324923
V. Inst. Count                           :        16268966755
V. Element Count                         :      4164804790539
V. Load Element Count                    :      1921282387058
FLOP Count                               :      1281280040326
MOPS                                     :         42021.575083
MOPS (Real)                              :        137499.248243
MFLOPS                                   :         11844.518710
MFLOPS (Real)                            :         38756.577193
A. V. Length                             :           255.996884
V. Op. Ratio (%)                         :            98.660787
L1 Cache Miss (sec)                      :             0.414586
CPU Port Conf. (sec)                     :             0.000000※1
V. Arith. Exec. (sec)                    :            35.594797※1
V. Load Exec. (sec)                      :            68.634671※1
VLD LLC Hit Element Ratio (%)            :             0.001852
FMA Element Count                        :       2002000063※1
Power Throttling (sec)                   :             0.000000※1
Thermal Throttling (sec)                 :             0.000000※1
Max Active Threads                       :                    4※2
Available CPU Cores                      :                    8※2
Average CPU Cores Used                   :             3.323850※2
Memory Size Used (MB)                    :          7884.000000
Non Swappable Memory Size Used (MB)      :           179.000000
```

時間情報

命令実行回数情報

ベクトル化情報・
メモリ情報・
並列化情報

プログラム情報 (2/4)

	項目	単位	説　明
1	Real Time	秒	経過時間
2	User Time	秒	ユーザ時間
3	Vector Time	秒	ベクトル命令実行時間
4	Inst. Count		全命令実行数
5	V. Inst. Count		ベクトル命令実行数
6	V. Element Count		ベクトル命令実行要素数
7	V. Load Element Count		ベクトル命令ロード要素数
8	FLOP Count		浮動小数点データ実行数
9	MOPS		ユーザ時間1秒あたりに実行された演算数(100万単位)
10	MOPS (Real)		経過時間1秒あたりに実行された演算数(100万単位)
11	MFLOPS		ユーザ時間1秒あたりに処理された浮動小数点データ実行要素数(100万単位)
12	MFLOPS (Real)		経過時間1秒あたりに処理された浮動小数点データ実行要素数(100万単位)
13	A. V. Length		平均ベクトル長
14	V. Op. Ratio	%	ベクトル演算率／ベクトル命令を使用して演算が行われた割合
15	L1 Cache Miss	秒	L1キャッシュミス時間
16	CPU Port Conf.	秒	CPUポート競合時間(※1)
17	V. Arith Exec.	秒	ベクトル命令実行時間(※1)
18	V. Load Exec.	秒	ベクトルロード実行時間(※1)
19	VLD LLC Hit Element Ratio	%	ベクトル命令によりロードされた要素のうち、LLCからロードされた要素の比率
20	FMA Element Count		FMA命令実行要素数(※1)
21	Power Throttling	秒	電力要因によるHW停止時間(※1)
22	Thermal Throttling	秒	温度要因によるHW停止時間(※1)
23	Max Active Threads		同時にアクティブだったスレッドの最大数(※2)
24	Available CPU cores		利用可能なCPUコアの個数(※2)
25	Average CPU Cores Used		平均CPUコア使用率(※2)
26	Memory Size Used	MByte	メモリ最大使用量
27	Non Swappable Memory Size Used	MByte	Partial Process Swapping機能でスワップアウトできないメモリの最大使用量

プログラム情報 (3/4)

- 実行時に環境変数**VE_PERF_MODE**に値を設定すると採取する性能カウンタが切り替わる
  “指定なし”または”**VECTOR-OP** “ ：主にベクトル演算に関連する情報
  “VECTOR-MEM” ：主にベクトルとメモリアクセスに関連する情報

```
 ********  Program  Information  ********
Real Time (sec)                          :           41.899895
User Time (sec)                          :          167.316072
Vector Time (sec)                        :           92.309990
Inst. Count                              :       175930159073
V. Inst. Count                           :        16278237491
V. Element Count                         :       564896066618
V. Load Element Count                    :       167991674972
FLOP Count                               :       432838957109
MOPS                                     :         4956.779900
MOPS (Real)                              :        19821.081454
MFLOPS                                   :         2583.413548
MFLOPS (Real)                            :        10330.507186
A. V. Length                             :           34.702533
V. Op. Ratio (%)                         :           80.776073
1 L1 I-Cache Miss (sec)                  :            0.256811
2 L1 O-Cache Miss (sec)                  :           32.683728
3 L2 Cache Miss (sec)                    :           32.884266
  FMA Element Count                      :       105937492420
4 Required B/F                           :            5.773960
5 Required Store B/F                     :            2.195626
6 Required Load B/F                      :            3.578334
7 Actual Load B/F                        :            0.404243
  Power Throttling (sec)                 :            0.000000
  Thermal Throttling (sec)               :            0.000000
  Max Active Threads                     :                   4
  Available CPU Cores                    :                   8
  Average CPU Cores Used                 :            3.993234
  Memory Size Used (MB)                  :         1028.000000
  Non Swappable Memory Size Used (MB)    :          179.000000
```

プログラム情報 (4/4)

	項目	単位	説　明
1	L1 I-Cache Miss	秒	L1 命令キャッシュミス時間
2	L1 O-Cache Miss	秒	L1 オペランドキャッシュミス時間
3	L2 Cache Miss	秒	L2 キャッシュミス時間
4	Required B/F		ロード命令とストア命令に指定されたバイト数から算出し た B/F (※1)
5	Required Store B/F		ストア命令に指定されたバイト数から算出した B/F （※1)
6	Required Load B/F		ロード命令に指定されたバイト数から算出した B/F （※1)
7	Actual V. Load B/F		ベクトルロード命令により実際に発生したメモリアクセス のバイト数から算出した B/F （※1)

(※1)：VE_PROGINF=DETAIL 時のみ出力、100以上の値は切り捨て

簡易性能解析(ftrace)(1/6)

```
$ nfort -ftrace a.f90 b.f90 c.f90
$ ./a.out （VE上での実行）
$ ls ftrace.out
ftrace.out
$ ftrace
*----------------------*
  FTRACE ANALYSIS LIST
*----------------------*

Execution Date : Thu Mar 22 17:32:54 2018 JST ①
Total CPU Time : 0:00'11"163 (11.163 sec.)    ②

③         ④                ⑤            ⑥      ⑦       ⑧     ⑨       ⑩      ⑪       ⑫       ⑬       ⑭
FREQUENCY  EXCLUSIVE        AVER.TIME     MOPS   MFLOPS  V.OP  AVER.    VECTOR L1CACHE CPU PORT VLD LLC PROC.NAME
           TIME[sec](  % )    [msec]                      RATIO V.LEN      TIME    MISS     CONF HIT E.%

    15000     4.762( 42.7)      0.317  77117.2  62034.6 99.45 251.0     4.605   0.002    0.000  100.00 FUNC_A
    15000     3.541( 31.7)      0.236  73510.3  56944.5 99.46 216.0     3.554   0.000    0.000  100.00 FUNC_B
    15000     2.726( 24.4)      0.182  71930.2  27556.5 99.43 230.8     2.725   0.000    0.000  100.00 FUNC_C
        1     0.134(  1.2)    133.700  60368.8  35641.2 98.53 214.9     0.118   0.000    0.000    0.00 MAIN
-------------------------------------------------------------------------------------------------------------
    45001    11.163(100.0)      0.248  74505.7  51683.9 99.44 233.5    11.002   0.002    0.000  100.00 total
```

(コンパイル、リンク時に**-ftrace**コンパイラオプションを指定)

(実行時、性能情報が格納されたftrace.outファイルを出力)

(**ftrace**コマンドで解析結果を表示)

※上記表示は環境変数VE_PERF_MODEが未設定またはVECTOR-OPを指定した例。

MPIプログラムのとき、性能情報が格納されたファイルが複数出力される。それらを**-f**オプションで指定する。

```
$ ls ftrace.out.*
ftrace.out.0.0   ftrace.out.0.1   ftrace.out.0.2   ftrace.out.0.3
$ ftrace -f ftrace.out.0.0 ftrace.out.0.1 ftrace.out.0.2 ftrace.out.0.3
```

簡易性能解析(ftrace)(2/6)

	項目	単位	説　明
①	Execution Date		実行終了時刻
②	Total CPU Time		全ての関数の CPU 時間の合計
③	FREQUENCY		関数の呼び出し回数
④	EXCLUSIVE TIME	秒(%)	関数の実行に要した EXCLUSIVE な CPU 時間と全関数の CPU 時間に対する比率
⑤	AVER.TIME	ミリ秒	関数の 1 回の実行に要した EXCLUSIVE な CPU 時間の平均
⑥	MOPS		“EXCLUSIVE TIME” 1 秒あたりに実行された演算数(100 万単位)
⑦	MFLOPS		“EXCLUSIVE TIME” 1 秒あたりに処理された浮動小数点 データ実行要素数(100 万単位)
⑧	V.OP RATIO	%	ベクトル演算率（ベクトル命令を使用して演算が行われた割合）
⑨	AVER.V.LEN		平均ベクトル長
⑩	VECTOR TIME	秒	ベクトル命令実行時間
⑪	L1CACHE MISS	秒	L1 キャッシュミス時間
⑫	CPU PORT CONF.	秒	CPU ポート競合時間
⑬	VLD LLC HIT E.%	%	ベクトル命令によりロードされた要素のうち、LLC からロードされた要素の比率
⑭	PROC.NAME		関数名

簡易性能解析(ftrace)(3/6)

- 環境変数VE_PERF_MODEが未設定またはVECTOR-OPが設定された場合は主にベクトル演算に関する項目が出力される
- VE_PERF_MODEにVECTOR-MEMが指定された場合は以下の出力がされる。この場合、主にベクトルとメモリアクセスの項目が出力される

```
*----------------------*
   FTRACE ANALYSIS LIST
*----------------------*

Execution Date : Sun Dec 15 21:51:48 2019 JST
Total CPU Time : 0:00'39"862 (39.862 sec.)

                                  ①        ②        ③        ④   ⑤     ⑥     ⑦         ⑧              ⑨
FREQUENCY  EXCLUSIVE        ...  L1ICACHE L1OCACHE L2CACHE    REQ. REQ.ST REQ.LD ACT.VLD     FLOP COUNT     FMA ELEM. PROC.NAME
           TIME[sec](  % )  ...       MISS     MISS    MISS    B/F    B/F    B/F     B/F                                

    15562    17.311( 43.4)  ...      0.005    4.441   4.442   5.12   1.34   3.78    0.10   16689871512    4486524600 funcA
    15562    17.235( 43.2)  ...      0.009    4.007   4.009   5.12   1.34   3.78    0.01   16689871512    4187422960 funcB
      253     1.886(  4.7)  ...      0.003    0.049   0.051   6.53   1.55   4.97    0.73   27492387186    9261379732 funcC
    15562     1.392(  3.5)  ...      0.010    0.013   0.021   6.77   3.42   3.35    0.14   21847989784    3469579024 fucnD
    15562     1.074(  2.7)  ...      0.006    0.073   0.077   6.85   3.53   3.32    0.13   19036465492    3469579024 funcE
  3147416     0.618(  1.6)  ...      0.000    0.030   0.000   8.39   2.41   5.98    0.00     387132168             0 funcF
        2     0.211(  0.5)  ...      0.000    0.000   0.000  39.45  21.54  17.91    0.00      46735724             0 funcG

                :

        1     0.000(  0.0)  ...      0.000    0.000   0.000   0.00   0.00   0.00    0.00             0             0 funcY
        2     0.000(  0.0)  ...      0.000    0.000   0.000   0.00   0.00   0.00    0.00             0             0 funcZ
--------------------------- ... ------------------------------------------------------------------------------------------
-
  3272238    39.862(100.0)  ...      0.038    8.620   8.620   6.26   2.29   3.97    0.30  103262749091   25237042300 total
```

簡易性能解析(ftrace)(4/6)

	項目	単位	説　明
①	L1ICACHE MISS	秒	L1 命令キャッシュミス時間
②	L1OCACHE MISS	秒	L1 オペランドキャッシュミス時間
③	L2CACHE MISS	秒	L2 キャッシュミス時間
④	REQ. B/F		ロード命令とストア命令に指定されたバイト数から算出し た B/F (*1)
⑤	REQ. ST B/F		ストア命令に指定されたバイト数から算出した B/F (*1)
⑥	REQ. LD B/F		ロード命令に指定されたバイト数から算出した B/F (*1)
⑦	ACT. VLD B/F		ベクトルロード命令により実際に発生したメモリアクセス のバイト数から算出した B/F (*1)
⑧	FLOP COUNT		浮動小数点データ実行要素数
⑨	FMA ELEM.		FMA 命令実行要素数

(*1) 100 以上の値は切り捨て

簡易性能解析(ftrace)(5/6)

- FTRACEでは、手続の入口/出口で性能情報を採取するため、手続の呼び出し回数が多いプログラムでプログラム全体の実行時間が増加してしまう。

```
$ nfort -ftrace -c a.f90
$ nfort -c main.f90 b.f90 c.f90
$ nfort -ftrace a.o main.o b.o c.o
$ ./a.out
```

- 目的の関数が含まれているファイルのみ**-ftrace**付きでコンパイルする
- リンク時にも**-ftrace**を指定する

- **-ftrace**なしでコンパイルされたファイル中の手続の性能情報は、それらを呼び出している手続の性能情報に含めて表示される
- システムライブラリ関数に関する性能情報
  - PROGINFで表示される性能情報には、プログラムから呼び出しているシステムライブラリ関数の性能情報も含まれる
  - FTRACEで表示される性能情報には、プログラムから呼び出しているシステムライブラリ関数の性能情報も含まれる。それらは、呼び出した手続の性能情報に含めて表示される

簡易性能解析(ftrace)(6/6)

ユーザ指定リージョン

- プログラムの局所的な部分の性能を知りたい場合に使用する

```
PROGRAM MAIN
PRINT*, "TEST"
CALL INIT
CALL FTRACE_REGION_BEGIN("U_REGION")
CALL SUB
CALL SUB
CALL FTRACE_REGION_END("U_REGION")
END
```

以下の手続きの実行で挟まれる範囲を測定可能

```
CHARACTER *(*) NAME
FTRACE_REGION_BEGIN(NAME)
FTRACE_REGION_END(NAME)
```

```
FREQUENCY  EXCLUSIVE       AVER.TIME    MOPS   MFLOPS V.OP  AVER.  …     PROG.UNIT
           TIME[sec](  % )    [msec]                  RATIO V.LEN
        2     1.539( 99.9)   769.251 31597.3 20799.5 99.83 250.0          sub
        1     0.001(  0.1)     0.868 13982.2     0.0 98.84 256.0          init
        1     0.000(  0.0)     0.160   272.9     0.2 76.53 250.9          main
-------------------------------------------------------------------------------
        4     1.540(100.0)   384.882 31584.1 20785.6 99.83 250.0          total

        1     1.539( 99.9)  1538.506 31597.2 20799.4 99.83 250.0          U_REGION
-------------------------------------------------------------------------------
```

演習問題 1

姫野ベンチマークのプログラムコードを用いて,コンパイル確認,実行確認,性能解析を行います.

① comp.shを用いてコンパイルし,run.shでジョブ投入してください.

$ cd practice_1

$ ./comp.sh

$ qsub run.sh

```
#!/bin/bash

nfort -report-format -report-diagnostics himeno.f90
rm -f *.mod *.o
```

```
#!/bin/bash
#PBS -q sx
#PBS -l elapstim_req=00:03:00
#PBS --venode 1
#PBS -v VE_PROGINF="DETAIL"

cd $PBS_O_WORKDIR
./a.out
```

演習問題１

② 実行が終了すると,run.sh.e.X(標準エラー出力)ファイルとrun.sh.o.X(標準出力)ファイルが作成されます．catコマンドを用いて結果を確認してください.

$ cat run.sh.e.X　(プログラム情報の表示)

$ cat run.sh.o.X　(実行結果の表示)

```
********  Program  Information  ********
 Real Time (sec)                        :           34.318045
 User Time (sec)                        :           34.312031
 Vector Time (sec)                      :            1.286641
 Inst. Count                            :        34534921414
 V. Inst. Count                         :          335019461
 V. Element Count                       :        85487941926
 V. Load Element Count                  :        39873331509
 FLOP Count                             :        42531814972
 MOPS                                   :         3681.994968
 MOPS (Real)                            :         3681.252872
 MFLOPS                                 :         1239.592460
 MFLOPS (Real)                          :         1239.342624
 A. V. Length                           :          255.173063
 V. Op. Ratio (%)                       :           72.928852
 L1 Cache Miss (sec)                    :            8.615777
 CPU Port Conf. (sec)                   :            0.000000
 V. Arith. Exec. (sec)                  :            0.528844
 V. Load Exec. (sec)                    :            0.720483
 VLD LLC Hit Element Ratio (%)          :           46.247477
 FMA Element Count                      :         6645555000
 Power Throttling (sec)                 :            0.000000
 Thermal Throttling (sec)               :            0.000000
 Memory Size Used (MB)                  :        15286.000000
```

```
  mimax= 1025  mjmax= 513  mkmax= 513
  imax= 1024  jmax= 512  kmax= 512
  Time measurement accuracy : .10000E-02
Now, start the actual measurement process.
The loop will be excuted in 5  times.
This will take about one minute.
Wait for a while.
  Loop executed for  5  times
  Gosa :   0.0000000
  MFLOPS:   1.3223826E+03   time(s):  34.1730000000000018
  Score based on Pentium III 600MHz :  15.9630928
```

演習問題１(つづき)

③ comp.shをエディタで編集し,-ftraceオプションを追加して,再度コンパイルしてジョブ投入してください．実行が終了したら,結果をftraceコマンドで表示してください.

$ vi(エディタ) comp.sh

$ ./comp.sh

$ qsub run.sh

$ ftrace -f ftrace.out

```
FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS  V.OP  AVER.    VECTOR L1CACHE CPU PORT VLD LLC PROC.NAME
           TIME[sec](  % )    [msec]                    RATIO V.LEN      TIME    MISS     CONF HIT E.%

       1    34.173( 99.6) 34173.230   3327.2   1244.6  70.14 255.5     1.150   8.615    0.000   46.25 JACOBI
       1     0.137(  0.4)   137.048  92163.3      1.9  98.06 253.3     0.137   0.000    0.000    0.00 INITMT
       1     0.000(  0.0)     0.204    369.9      1.5   0.30 112.0     0.000   0.000    0.000  100.00 HIMENOBMTXP_F90
       1     0.000(  0.0)     0.058   8264.3      0.0  96.78 255.7     0.000   0.000    0.000    0.00 INITMEM
       1     0.000(  0.0)     0.011    581.2      0.0   0.00   0.0     0.000   0.000    0.000    0.00 GRID_SET
       2     0.000(  0.0)     0.002    256.5      0.0   0.00   0.0     0.000   0.000    0.000    0.00 SECOND
       1     0.000(  0.0)     0.000   1071.3      0.0   0.00   0.0     0.000   0.000    0.000    0.00 READPARAM
-------------------------------------------------------------------------------------------------------------
       8    34.311(100.0)  4288.819   3682.0   1239.6  72.93 255.2     1.287   8.615    0.000   46.25 total
```

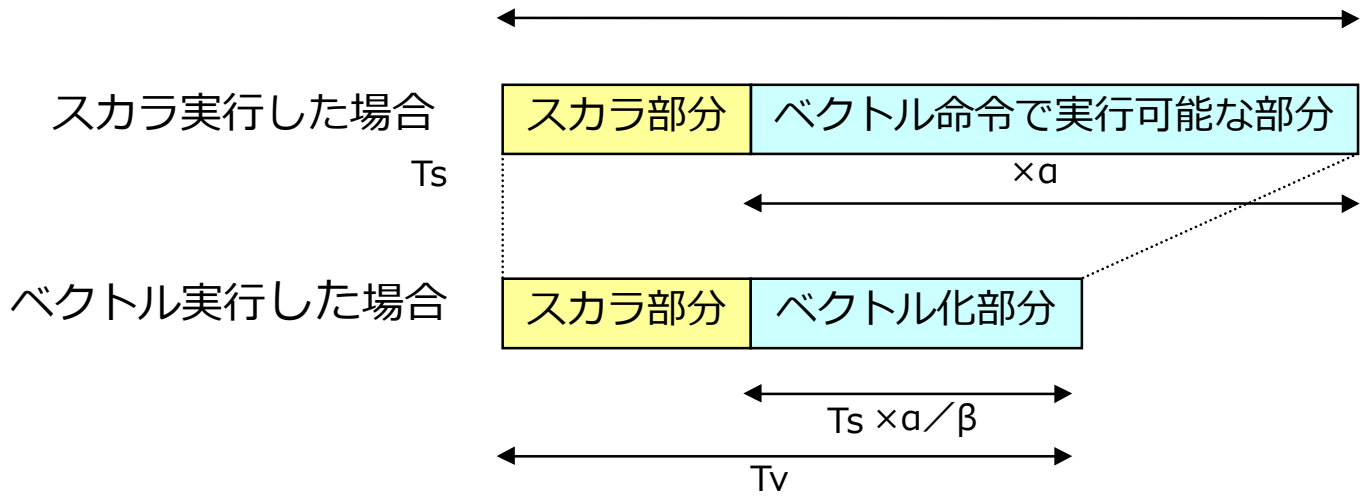
3. SX-Aurora TSUBASAチューニングのポイント

SX-Aurora TSUBASAチューニングのポイント

- ベクトル型スーパーコンピュータであるSX-Aurora TSUBASAのチューニングのポイントは以下の３点.
  1. ベクトル化率(ベクトル演算率)の向上
  2. ループ長の拡大
  3. メモリアクセスの効率化

- また並列実行時には以下の３点を改善することで並列化の台数効果の向上を図る.
  1. 並列化率の向上(非並列化部分を最小限)
  2. 負荷バランスの均衡
  3. オーバーヘッドの削減

ベクトル化率とは



α：ベクトル化率

β：ベクトル命令の性能向上比

Ts　：スカラ実行したときの実行時間

Tv　：ベクトル実行したときの実行時間

性能向上比＝Ｔｓ／Ｔｖ＝β／　（α　*(1−β　)＋β　）

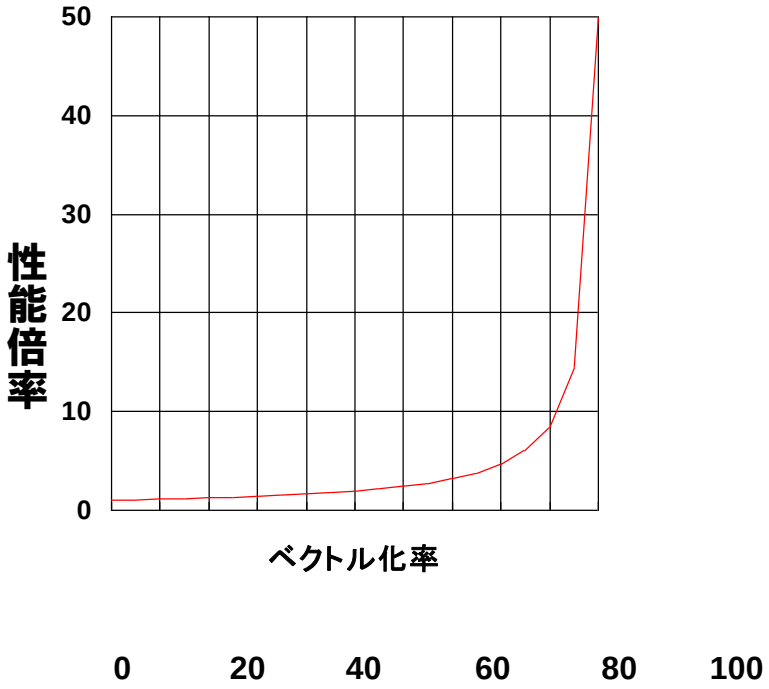
ベクトル演算率＝　（Ｔｓ*α／β　　）／Ｔｖ

=α／　（α　*(1−β　)＋β　）

ベクトル化率を正確に求めることは困難であるため，
プログラム情報(proginf)に表示されるベクトル演算率
（ベクトル演算が実行された割合）で代用する．

ベクトル化率と性能向上

ベクトル化率が十分に高くなって初めて効果発揮



ベクトル化による性能向上比を50倍と仮定
　ベクトル化率　50%，全体の性能は2倍
　ベクトル化率　80%でも，全体の性能は4.6倍にしかならない．

ベクトル化で十分な高速化を行うためには，
ベクトル化率を可能な限り*100%*に近づける

※上記グラフはアムダール則に基づき，ベクトル化率V=100%時の性能向上を５０倍として算出．

$$性能倍率 = \frac{1}{(1-V) + \frac{V}{50}}$$

ベクトル化率の向上

- コンパイル時に「編集リスト」と「ベクトル化診断メッセージ」を採取.
  - コンパイルオプションに 「-report-all –fdiag-vector=2」を付与
- ベクトル化が出来ていない箇所を特定し，ベクトル化不可の理由を確認.

```
  7: +------>        do i=1,n
  8: |                 a(i) = a(i) * b(i) - c(i)
  9: |                 if(a(i).le.0.d0) write(6,*) a(i)
 10: +------         enddo

7: vec( 103): Unvectorized loop.
7: vec( 180): I/O statement obstructs vectorization.
9: opt(1118): This I/O statement inhibits optimization of loop..
```

※ベクトル化を阻害する文(write)がある

```
 12: +------>        do i=2,n
 13: |                 a(i) = a(i-1) * b(i) - a(i-1) * c(i)
 14: +------         enddo

12: vec( 103): Unvectorized loop.
12: vec( 113): Overhead of loop division is too large.
13: opt(1037): Feedback of array elements.
13: vec( 120): Unvectorizable dependency.: A
```

※配列aに対するアクセスにベクトル化を阻害する依存関係がある

```
 16: +------>        do i=1,n
 17: |                 call sub(b)
 18: |                 a(i) = a(i) * b(i) - c(i)
 19: +------         enddo

16: vec( 103): Unvectorized loop.
16: vec( 110): Vectorization obstructive procedure reference.:
SUB
17: opt(1025): Reference to this procedure inhibits
optimization.: SUB
```

※ベクトル化を阻害するサブルーチンcallがある

```
 23: S------>        do i=1,n
 24: |                 a(itbl(i)) = a(itbl(i)) + b(i) * c(i)
 25: S------         enddo

23: vec( 102): Partially vectorized loop.
24: opt(1036): Potential feedback - use directive or compiler
option if OK.
24: vec( 122): Dependency unknown. Unvectorizable dependency is
assumed.: A
```

※配列aに対するアクセスに重なりがある(依存関係があるか)かどうか判断できない

- コンパイルオプションの追加，ベクトル化指示行の挿入，ソースコードの修正などによりベクトル化を促進.

ループ中のIF文の最適化

- ループ中にIF文があり，IF文が真になる場合にベクトル化不可の処理が実行される場合，あるいはIF文が真になる場合にベクトル化可能な処理が実行されるがIF文真率が低い場合の最適化の方法について

```
 9: S------>          do i=1,n
10: |       F           a(i) = a(i) * b(i) - c(i)
11: |                     if(a(i).le.0.d0) a(i)=1/a(i-1)
12: S------          enddo
```

- IF文が真になる回数をカウントし，真になる場合のループ変数を作業配列に格納しておく．ループの実行後にIF文が真になる回数だけ繰り返す処理を追加する(但し，IF文の真率が高いと効果が得られない可能性が高い).

```
 9: V------>          do i=1,n
10: |       F           a(i) = a(i) * b(i) - c(i)
11: |                     if(a(i).le.0.d0) then
12: |                       icnt=icnt+1
13: |                       itbl(icnt)=i
14: |                     endif
15: V------          enddo
16: S------>         do i=1,icnt
17: |                 a(itbl(i))=1/a(itbl(i-1))
18: S------          enddo
```

ベクトルで処理

スカラで処理

ソースプログラムの変更による依存関係の回避 (1/3)

ベクトル化できないループ

```
DO I=1, N
  IF(X(I).LT.S) THEN
    T = X(I)
  ELSE IF(X(I).GE.S) THEN
    T = -X(I)
  ENDIF
  Y(I) = T
ENDDO
```

Tは定義されないかもしれない

Tが参照前に必ず定義されるように変形する

ベクトル化可能なループ

```
DO I=1, N
  IF(X(I).LT.S) THEN
    T = X(I)
  ELSE  ! IF(X(I).GE.S) THEN
    T = -X(I)
  ENDIF
  Y(I) = T
ENDDO
```

Tは必ず定義される

-fdiag-vector=2 を指定することにより，以下のメッセージが出力される

```
nfort: vec( 103): test.f, line 3: Unvectorized loop.
nfort: vec( 113): test.f, line 3: Overhead of loop division is too large.
nfort: opt(1019): test.f, line 5: Feedback of scalar value from one loop pass to another.: T
nfort: vec( 121): test.f, line 5: Unvectorizable dependency.
nfort: vec( 121): test.f, line 7: Unvectorizable dependency.
```

ソースプログラムの変更による依存関係の回避 (2/3)

ソースプログラム

```
DO I=1, N
  IF(A(I).GT.0.0) THEN
      S = S + B(I)
   ELSE
      S = S + C(I)
   END IF
ENDDO
```

```
DO I=1, N
  IF(A(I).GT.0.0) THEN
      T = B(I)
   ELSE
      T = C(I)
   END IF
   S = S + T
ENDDO
```

S　は繰り返し間にまたがって定義・　参照されるため，ベクトル化できない

ソースを書き換えることにより
総和型のマクロ演算に適合するので
ベクトル化される

ソースプログラムの変更による依存関係の回避 (3/3)

ソースプログラム

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

X(I) = X(I-1)....にベクトル化を阻害する依存関係があるため，ベクトル化できない

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

ソースを変形すると，漸化式型のマクロ演算
X(I) = 式 ± X(I-1) ＊ 式
に適合するのでベクトル化できる

手続きのインライン展開

手続きのインライン展開による改善

```
DO I=1, N
   A(I)=FUN(B(I),C(I))+D(I)
ENDDO
```

```
FUNCTION FUN(X, Y)
     FUN = SQRT(X) * Y
END FUNCTION FUN
```

```
nfort: vec( 103): test.f, line 3: Unvectorized loop.
nfort: vec( 110): test.f, line 3: Vectorization obstructive procedure reference.: FUN
nfort: opt(1025): test.f, line 4: Reference to this procedure inhibits optimization.: FUN
```

コンパイル時オプション -finline-functionsを指定することにより，FUNがインライン展開され上記ループはベクトル化される

-finline-functions指定時のコンパイラの変形イメージ

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

```
nfort: vec( 101): test.f, line 3: Vectorized loop.
nfort: inl(1222): test.f, line 4: Inlined: FUN
```

配列の重なりがない場合(IVDEP指示行)

IVDEP指示行

ソースプログラム

```
DO I=1, N
    A(IX(I)) = A(IX(I)) + B(I)
ENDDO
```

A(IX(I))の依存関係不明でベクトル化されない.
-mlist-vector を指定することにより，ベクトル化を行うこともできるが…

-fdiag-vector=2 を指定することにより，以下のメッセージが出力される

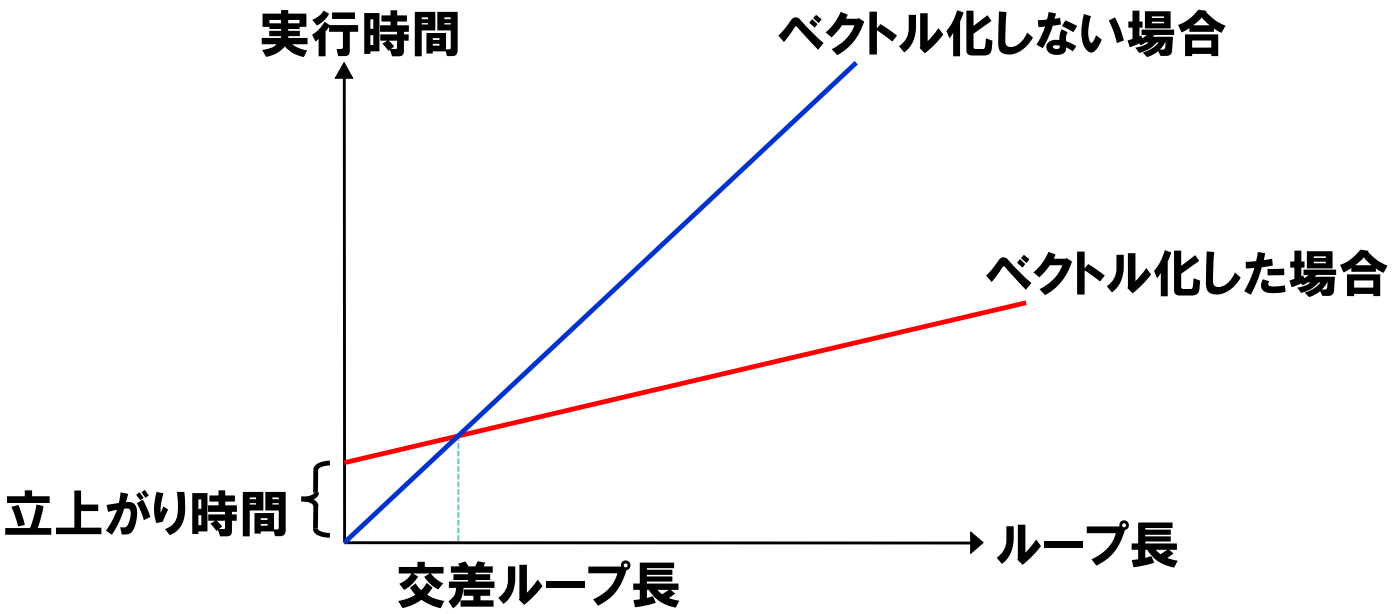
```
nfort: vec( 101): test.f, line 5: Vectorized loop.
nfort: opt(1036): test.f, line 6: Potential feedback - use directive or compiler option if OK.
nfort: vec( 126): test.f, line 6: Idiom detected.: LIST VECTOR
```

```
!NEC$ IVDEP
DO I=1, N
    A(IX(I))=A(IX(I))+B(I)
ENDDO
```

配列IX(I)の値に，重複した要素のないことがわかっているならば指示行IVDEPを挿入することで高速にベクトル化できる
（例： IX(I)が，9,3,2,4,1,5,7,10,8,…）

※配列IX(I)の値に，重複があるのにivdep指示行を指定すると不正な結果になる.

ループ長（ベクトル長）



ベクトル処理が開始されるまでに少し時間がかかる．（立ち上がり時間）
そのためループ長が非常に短い場合，ベクトル化しない方が速い．
(交差ループ長 3 程度)
ループ長をできるだけ長くすることで，ベクトル化による高速化の効果が大きいことがわかる．

指示行の挿入によるループ長の拡大 (1/2)

- SX-Aurora TSUBASAでは長ベクトル・ストライドアクセスのループより，短ベクトル・連続アクセスのループの方が処理時間が短い場合がある.
- SX-Aurora TSUBASAのコンパイラは，ループ長より連続アクセスを優先するようにループの入れ替えを行う(または行わない).

LOOP1
- 長ベクトル
  →ループ長25600
- ストライドアクセス
  →65要素飛びアクセス

```
27: +------>          do i = 1,64
28: |V----->            do j = 1,25600
29: ||      F             d1(i,j) = d1(i,j)+a1(i,j)+b1(i,j)*c1(i,j)
30: |V-----             enddo
31: +------           enddo
```

LOOP2
- 短ベクトル
  →ループ長64
- 連続アクセス

```
35:           !NEC$ interchange
36: X------>          do i = 1,64
37: |*----->            do j = 1,25600
38: ||      F             d1(i,j) = d1(i,j)+a1(i,j)+b1(i,j)*c1(i,j)
39: |*-----             enddo
40: X------           enddo
```

ストライドアクセスの場合，
バンク競合時間が増大

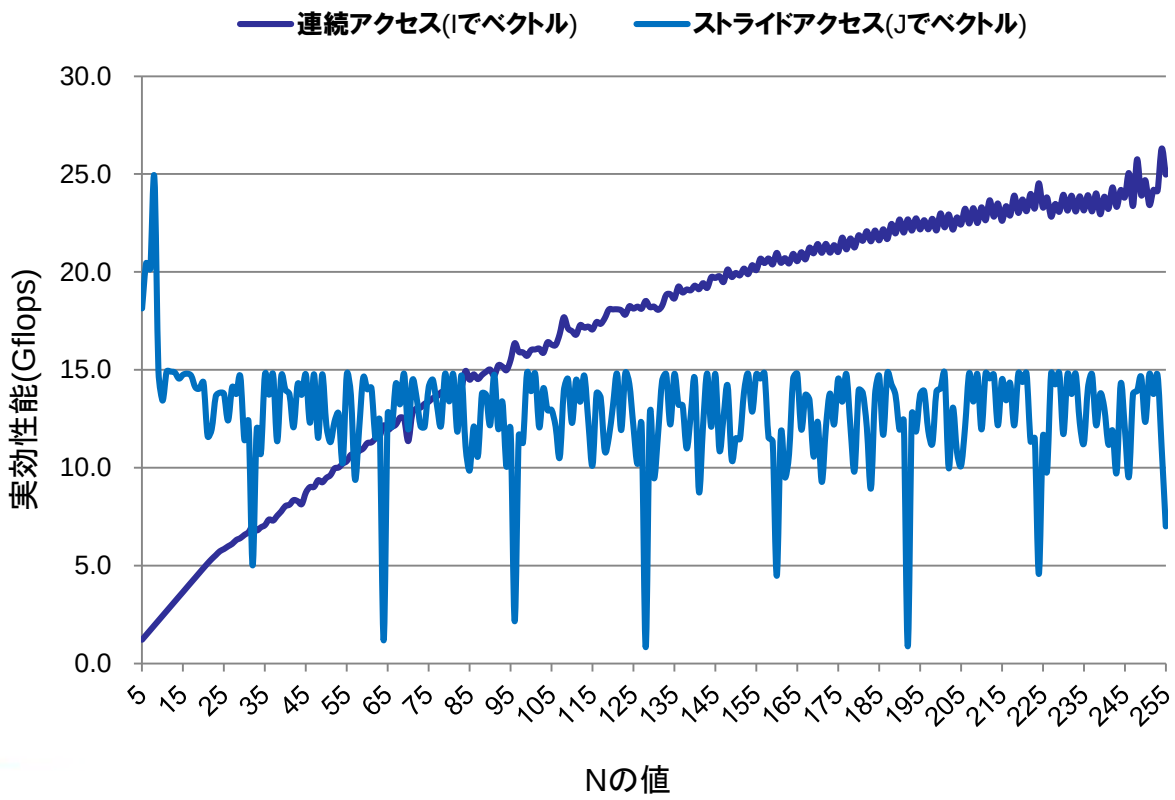
FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	L1CACHE MISS	CPU PORT CONF	VLD LLC HIT E.%	PROC.NAME
20000	82.686(90.5)	4.134	3192.4	1188.9	99.31	256.0	82.723	0.001	19.394	0.00	LOOP1
20000	8.659(9.5)	0.433	31397.8	11352.8	96.42	64.0	8.659	0.001	0.000	0.00	LOOP2

指示行の挿入によるループ長の拡大 (2/2)

SX-Aurora TSUBASAで連続アクセス，ストライドアクセスの性能比較

```
36: +------>          do j = 1,25600
37: |V----->            do i = 1,N
38: ||       F             d1(i,j) = d1(i,j)+a1(i,j)+b1(i,j)*c1(i,j)
39: |V-----             enddo
40: +------           enddo
```

Nを5～255とした場合にiのループでベクトル化(連続アクセス)した場合とjのループでベクトル化(ストライドアクセス)した場合の実効性能(Gflops)を比較



SX-Aurora TSUBASAではループ長がXX以上であれば連続アクセスとした方が実効性能が高い

演習問題２

演習問題１のプログラム性能を確認します.

① 演習問題1のFtrace結果と編集リストを確認してください.

$ cd practice_1

$ cat himeno.L

```
  250: +------>    do loop=1,nn
  251: |+----->       do k=2,kmax-1
  252: ||+---->          do j=2,jmax-1
  253: |||S--->             do i=2,imax-1
  254: ||||   F                s0=a(I,J,K,1)*p(I+1,J,K) &
   :
  262: ||||                                    -p(I+1,J,K-1)+p(I-1,J,K-1)) &
  263: ||||                         +c(I,J,K,1)*p(I-1,J,K) &
  264: ||||                         +c(I,J,K,2)*p(I,J-1,K) &
  265: ||||                         +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
  266: ||||                      ss=(s0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
  267: ||||                      p(I,J,K)=p(I,J,K)+OMEGA *SS
  268: |||S---                enddo
  269: ||+----              enddo
  270: |+-----           enddo
  271: +------        enddo
```

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	L1CACHE MISS	CPU PORT CONF	VLD LLC HIT	PROC.NAME E.%
1	34.173(99.6)	34173.230	3327.2	1244.6	70.14	255.5	1.150	8.615	0.000	46.25	JACOBI

演習問題２

演習問題１で用いた姫野ベンチマークのプログラムコードを修正しチューニング効果を確認します.

② himeno.f90をエディタで編集し,プログラムコードを修正してください.

$ cd practice_2

$ vi(エディタ) himeno.f90

```
250: +------>     do loop=1,nn
251: |+----->        do k=2,kmax-1
252: ||+---->           do j=2,jmax-1
253: |||V--->              do i=2,imax-1
254: ||||    F                s0=a(I,J,K,1)*p(I+1,J,K) &
 :
266: ||||    F                ss=(s0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
267: ||||    F                wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
268: |||V---                enddo
269: ||+-----            enddo
270: |+------         enddo
271: |          !
272: |V=====>        p(2:imax-1,2:jmax-1,2:kmax-1)= &
273: |                    wrk2(2:imax-1,2:jmax-1,2:kmax-1)
274: |          !
275: +------      enddo
```

修正

追加

演習問題２

- プログラムコードの修正前後の編集リストを確認し,ベクトル化が行われていることを確認します.

```
  250: +------>    do loop=1,nn
  251: |+----->    do k=2,kmax-1
  252: ||+---->    do j=2,jmax-1
  253: |||S--->    do i=2,imax-1
  254: ||||    F      s0=a(I,J,K,1)*p(I+1,J,K) &
  255: ||||                +a(I,J,K,2)*p(I,J+1,K) &
  256: ||||                +a(I,J,K,3)*p(I,J,K+1) &
  257: ||||                +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K) &
  258: ||||                           -p(I-1,J+1,K)+p(I-1,J-1,K)) &
  259: ||||                +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1) &
  260: ||||                           -p(I,J+1,K-1)+p(I,J-1,K-1)) &
  261: ||||                +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1) &
  262: ||||                           -p(I+1,J,K-1)+p(I-1,J,K-1)) &
  263: ||||                +c(I,J,K,1)*p(I-1,J,K) &
  264: ||||                +c(I,J,K,2)*p(I,J-1,K) &
  265: ||||                +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
  266: ||||           ss=(s0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
  267: ||||           p(I,J,K)=p(I,J,K)+OMEGA *SS
  268: |||S---      enddo
  269: ||+----      enddo
  270: |+-----      enddo
  271: +------      enddo
```

```
  250: +------>    do loop=1,nn
  251: |+----->    do k=2,kmax-1
  252: ||+---->    do j=2,jmax-1
  253: |||V--->    do i=2,imax-1
  254: ||||    F      s0=a(I,J,K,1)*p(I+1,J,K) &
  255: ||||                +a(I,J,K,2)*p(I,J+1,K) &
  256: ||||                +a(I,J,K,3)*p(I,J,K+1) &
  257: ||||                +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K) &
  258: ||||                           -p(I-1,J+1,K)+p(I-1,J-1,K)) &
  259: ||||                +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1) &
  260: ||||                           -p(I,J+1,K-1)+p(I,J-1,K-1)) &
  261: ||||                +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1) &
  262: ||||                           -p(I+1,J,K-1)+p(I-1,J,K-1)) &
  263: ||||                +c(I,J,K,1)*p(I-1,J,K) &
  264: ||||                +c(I,J,K,2)*p(I,J-1,K) &
  265: ||||                +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
  266: ||||    F      ss=(s0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
  267: ||||    F      wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
  268: |||V---      enddo
  269: ||+----      enddo
  270: |+-----      enddo
  271: |          !
  272: |V=====>    p(2:imax-1,2:jmax-1,2:kmax-1)= &
  273: |              wrk2(2:imax-1,2:jmax-1,2:kmax-1)
  274: |          !
  275: +------      enddo
```

演習問題２(つづき)

③ comp.shは-ftraceオプションが指定されていることを確認し再度コンパイルしてジョブ投入してください．実行が終了したら結果をftraceコマンドで表示してください．

$ ./comp.sh

$ qsub run.sh

$ ftrace -f ftrace.out

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	L1CACHE MISS	CPU PORT CONF	VLD LLC HIT E.%	PROC.NAME
1	0.963(87.5)	962.863	91699.4	44172.0	99.35	255.5	0.963	0.000	0.000	46.93	JACOBI
1	0.137(12.5)	137.120	92114.8	1.9	98.06	253.3	0.137	0.000	0.000	0.00	INITMT
1	0.000(0.0)	0.201	379.0	1.8	0.29	112.0	0.000	0.000	0.000	100.00	HIMENOBMTXP_F90
1	0.000(0.0)	0.058	8256.4	0.0	96.78	255.7	0.000	0.000	0.000	0.00	INITMEM
1	0.000(0.0)	0.012	557.2	0.0	0.00	0.0	0.000	0.000	0.000	0.00	GRID_SET
2	0.000(0.0)	0.002	275.3	0.0	0.00	0.0	0.000	0.000	0.000	0.00	SECOND
1	0.000(0.0)	0.000	1028.8	0.0	0.00	0.0	0.000	0.000	0.000	0.00	READPARAM
8	1.100(100.0)	137.532	91728.8	38656.2	99.19	255.2	1.100	0.000	0.000	46.93	total

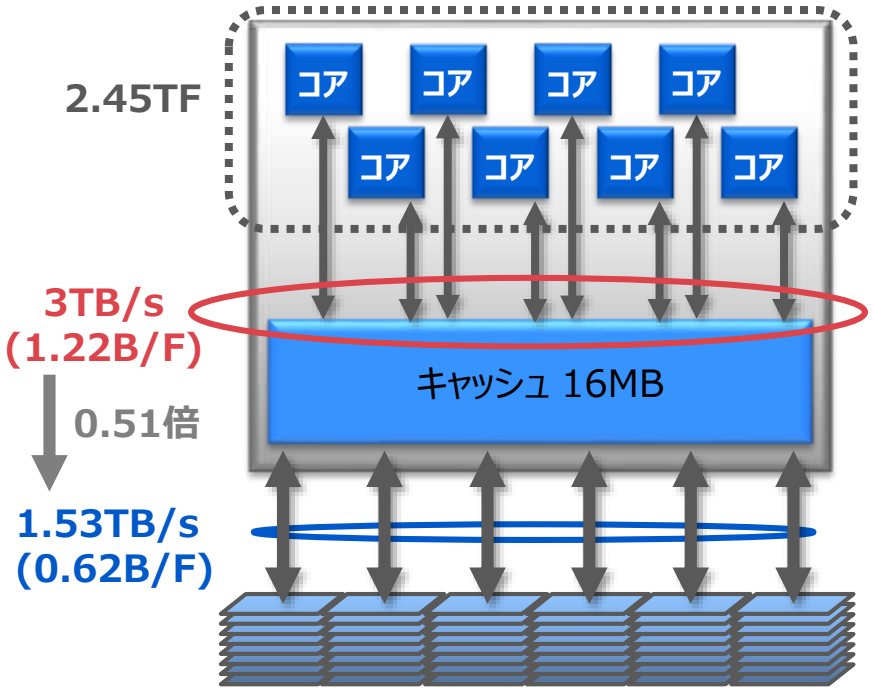
Byte/Flop値を用いた性能分析

- プログラムの性能特性を調査する際に、1つの演算を処理するために必要なメモリアクセス量を示す指標として、Byte/Flopがある
  - 例：A(I)=B(I)+C(I)の場合、(3op x 8B)/1演算= 24B/F
- SX-Aurora TSUBASAでは、2種類のByte/Flop値を使用することにより、プログラムの性能特性を調査することが可能
  - **Required B/F(REQ. B/F) → LLCの負荷状況を見る**
    - コンパイル後の命令列上のメモリアクセス要素数をカウント
    - ストライドアクセスや、LLCヒットなどは考慮していない
  - **Actual B/F(ACT. B/F) → メモリの負荷状況をみる**
    - LLCヒットした要素はカウントせず、実際にメモリをアクセスしたロードリクエスト数をカウント
      - ストアデータを実際にメモリに書き出すタイミングはわからないが、REQ. ST B/F値が大きい場合は(ACT.B/F＋REQ.ST B･F)で考える
    - ストライドアクセスなどを考慮
  - 例：ループ長=100、ストライド=16Bのロード処理を実行する場合（ブロックロードの単位：128B)
    - ロード要素数：100
    - 実際にメモリをアクセスするリクエスト数：200
  - プログラムを実行した際に求められるREQ.B/F、ACT.B/Fと、ハードウェアのLLCのB/FとMemoryのB/Fを比較することで、そのプログラムが演算性能依存、LLC BW依存、Memory BW依存であるのかを大まかに決めることができる
    - 本指標にはレイテンシの概念が含まれないので、メモリレイテンシ依存のプログラムはB/F値では判断できない

ハードウェアのByte/Flopとプログラムの Byte/Flopの関係

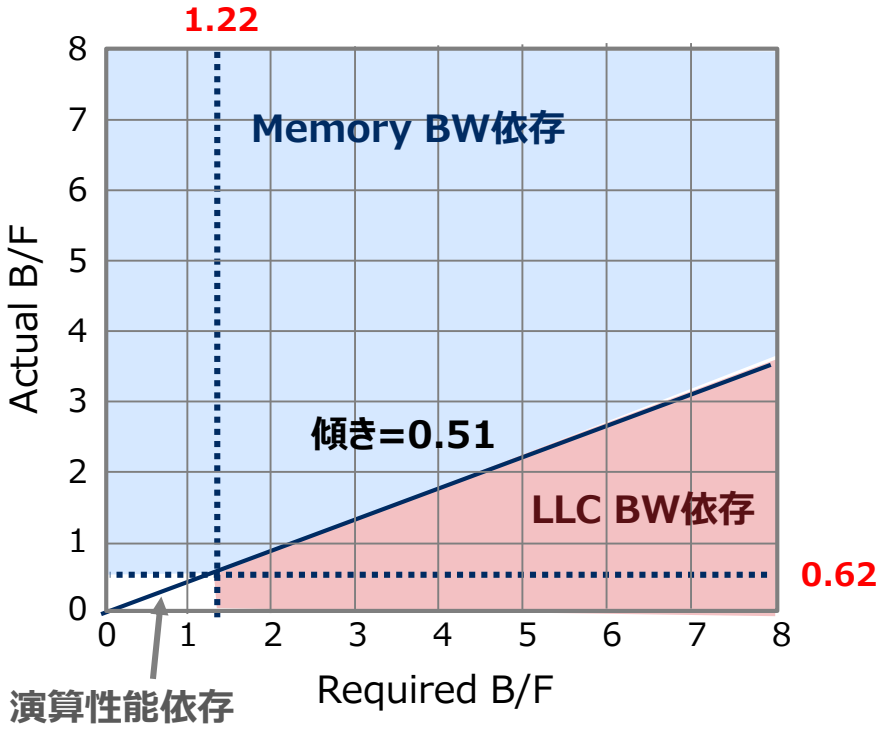
- ハードウェアのもつByte/Flopは、**LLC : 1.22B/F**、**メモリ:0.62B/F**である（左下図）
- 実際にプログラムを実行した際のLLC負荷度合いはREQ.B/Fを、メモリ負荷度合いはACT.B/Fを参考にする
  - REQ.B/F値がハードウェアのLLC B/Fの1.22以上の場合はLLC BW依存と考えられる
  - ACT.B/F値がハードウェアのメモリ B/Fの0.62以上の場合はMem BW依存と考えられる
  - REQ.B/F値が1.22以上、かつACT.B/F値が0.62以上の場合は右下グラフのように両者の比率に応じて依存が変わる

【ハードウェアのByte/Flop】



※HWのB/F値は、全演算命令がFMA演算になっているものと仮定

【プログラムのByte/Flop値と性能依存の関係】



Byte/Flop値の採取方法

- Req.B/F、Act.B/Fを採取するためには、実行時に以下の環境変数を設定する

  （本環境変数を指定するとLLC ヒット率が出力されなくなる）

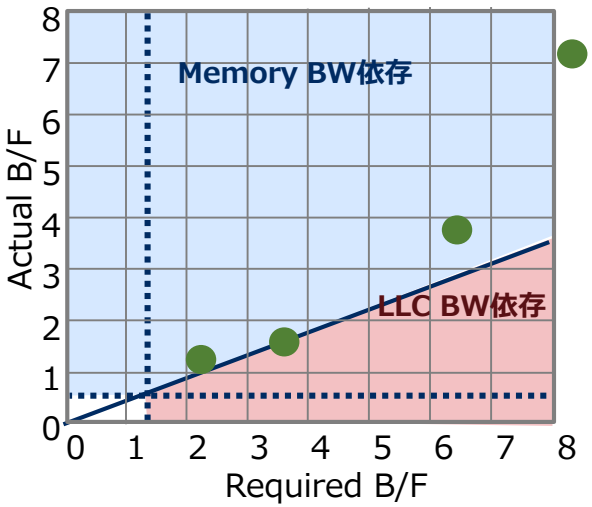
  export VE_PERF_MODE = VECTOR-MEM

- 以下のような性能解析ツール(FTRACE)の結果が得られる

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER. TIME [msec]	MOPS	MFLOPS	V. OP RATIO	AVER. V. LEN	VECTOR TIME	L1ICACHE MISS	L1OCACHE MISS	L2CACHE MISS	REQ. B/F	REQ. ST B/F	REQ. LD B/F	ACT. VLD B/F	FLOP COUNT	FMA ELEM.	PROC. NAME
10916640	3689.111(9.0)	0.338	33727.9	16248.2	99.70	246.9	3663.974	6.912	16.972	23.111	3.67	0.08	3.59	1.63	59941561006080	0	FuncA
49925	1538.149(3.8)	30.809	55018.8	28149.4	99.31	246.5	1537.069	0.541	0.433	0.933	6.32	2.63	3.69	1.18	43297929830325	3456989020800	FuncB
720917	1326.677(3.2)	1.840	35626.2	14895.6	98.07	237.0	975.714	64.377	176.536	230.865	8.45	4.18	4.27	3.08	19761668429465	5645730682926	FuncC
2162751	1157.990(2.8)	0.535	56994.1	37313.0	99.33	247.0	1040.146	38.159	71.358	108.008	2.21	0.86	1.35	0.29	43208102168538	13513161692760	FuncD

REQ. B/F	REQ. ST B/F	REQ. LD B/F	ACT. VLD B/F	FLOP COUNT	FMA ELEM.	PROC. NAME
3.67	0.08	3.59	1.63	59941561006080	0	FuncA
6.32	2.63	3.69	1.18	43297929830325	3456989020800	FuncB
8.45	4.18	4.27	3.08	19761668429465	5645730682926	FuncC
2.21	0.86	1.35	0.29	43208102168538	13513161692760	FuncD

※Actual B/Fは、(ACT.VLD B/F＋REQ.ST B/F)でプロット



メモリアクセスと性能の関係

LLCアクセスは、128Bを１とした場合に２のべき乗ストライドの場合に性能が落ちる

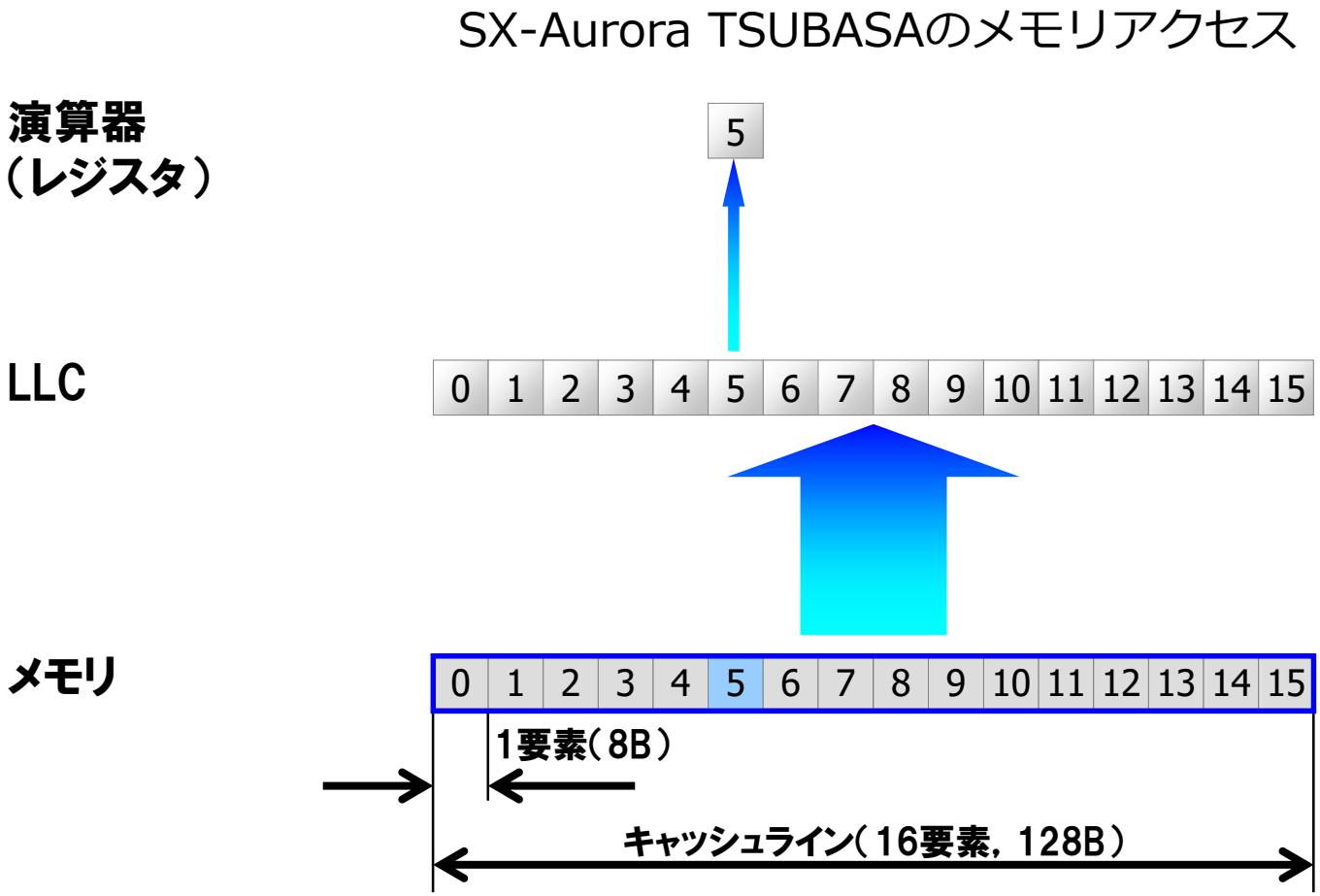
- 256B(128x2)飛びの場合は最速時の1/2、512B(128x4)飛びの場合は最速時の1/4の性能、2048B(128x16)飛び以上の場合は最速時の1/16に落ちる
- パディングなどで2のべき乗ストライドを避けることで性能低下を抑止

メモリ(HBM)アクセスは、128B x 3のストライド時にバンク競合の影響が出やすい

- メモリBWを最大限の性能を実現するため、HBMを6個搭載
- メモリのバンク数は128B x 1536個(=512x3)を有しているが、2のべき乗数にはなっていない
- メモリバンク競合を回避するためには128Bx1536飛びアクセスを避ける方が良い
- ただし、メモリバンク競合が発生した場合はHWが他のRQを先に処理するなどして高速化の工夫は施してある

メモリアクセス単位について(1/2)

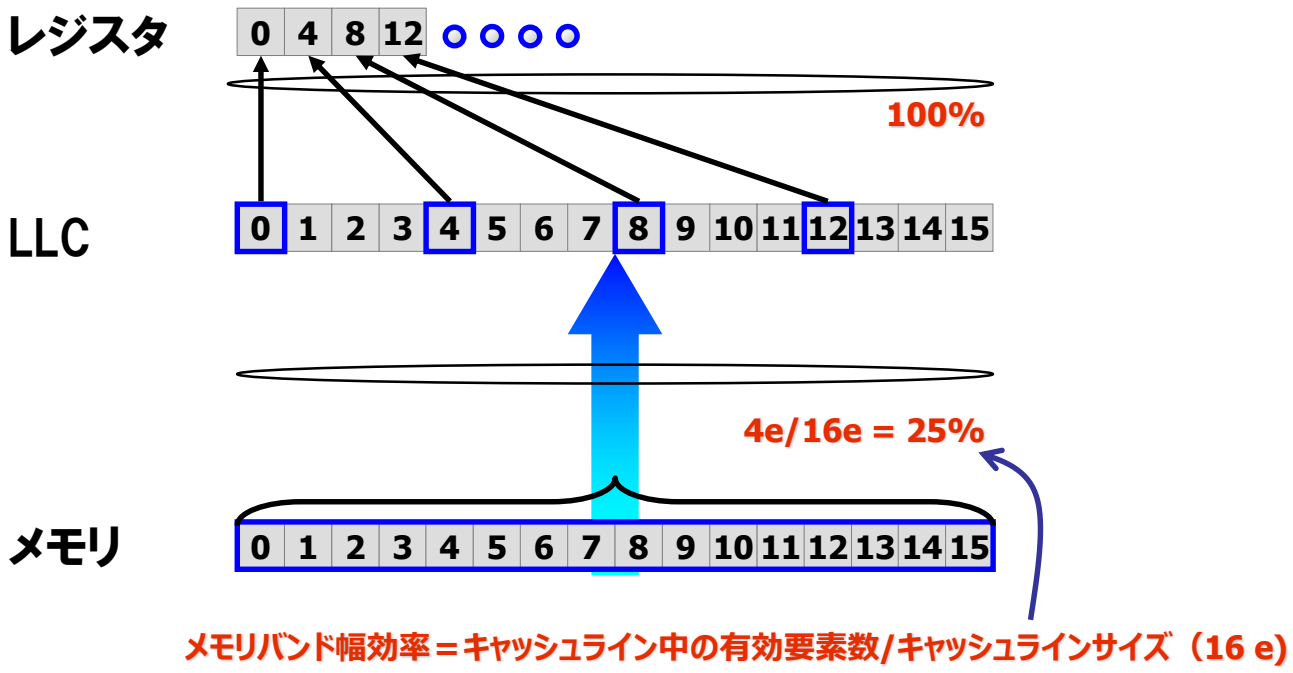
SX-Aurora TSUBASAはメモリアクセス単位が128B



メモリアクセス単位について(2/2)

ストライドアクセスの場合（例：４要素飛びアクセス）
：不要な要素(以下の例では要素0,4,8,12以外)を含む128Bをロード
※以下の例では４倍のデータをメモリからロードする

SX-Aurora TSUBASAのメモリアクセス



演習問題３

演習用のプログラムコードを用いて,演算性能依存,Memory BW依存,LLC BW依存について性能解析を行います.

① 各プログラムコードについてcomp.shを用いてコンパイルしてください.

$ cd practice_3/CAL　（演算性能依存）
$ cd practice_3/MEM　（Memory BW依存）
$ cd practice_3/LLC　（LLC BW依存）

② 各プログラムコードをrun.shでジョブ投入してください．実行が終了したら結果をftraceコマンドで表示してください.

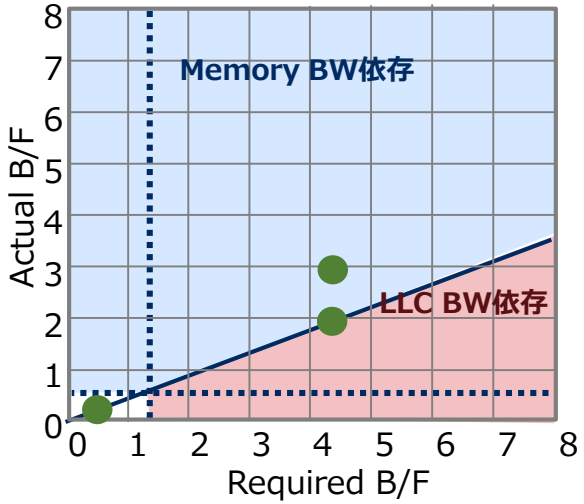
$ ./comp.sh
$ qsub run.sh
$ ftrace -f ftrace.out

演習問題３

③ ftraceコマンドによる結果を確認すると以下のような値になることを確認してください.

	EXCLUSIVE TIME	Required B/F	Required Store B/F	Actual V. Load B/F	Actual B/F
演算性能依存	8.75	0.39	0.07	0.26	0.33
Memory BW依存	17.46	4.25	0.25	2.75	3.00
LLC BW依存	11.70	4.26	0.26	1.78	2.04

※Actual B/F=(ACT.VLD B/F＋REQ.ST B/F)



演習問題３(つづき)

④ 演算性能依存のプログラムコード（himeno.f90）をエディタで編集し，プログラムコードを修正してください.

$ cd practice_3/CAL

$ vi(エディタ) himeno.f90

```
249:              real(4) :: ww(2:imax-1,2:kmax-1)        ← 追加
  :
252: +------>     do loop=1,nn
253: |V=====>        ww(2:imax-1,2:kmax-1) = sqrt(wrk1(2:imax-1,1,2:kmax-1))
254: |+----->        do k=2,kmax-1                          ← 追加
255: ||+---->           do j=2,jmax-1
256: |||V--->              do i=2,imax-1
257: ||||                    aa = a(I,J,K,1)
258: ||||                    bb = b(I,J,K,1)
  :
265: ||||                      +sqrt(cc*pp)**3 &
266: ||||                      +ww(I,K)        ← 修正
267: ||||    F                 p(I,J,K)=p(I,J,K)+OMEGA*SO
268: |||V---                 enddo
269: ||+----              enddo
270: |+-----          enddo
271: +------       enddo
```

演習問題３(つづき)

⑤ Memory BW依存およびLLC BW依存のプログラムコード（himeno.f90）をエディタで編集し,プログラムコードを修正してください.

$ cd practice_3/MEM or % cd practice_3/LLC

$ vi(エディタ) himeno.f90

```
254: +------>     do loop=1,nn
255: |+----->        do k=2,kmax-1
256: ||          !nec$ outerloop_unroll(4)        <- 追加
257: ||U---->             do j=2,jmax-1
258: |||V--->                  do i=2,imax-1
259: ||||    F                    s0=a(I,J,K,1)*p(I+1,J,K) &
  :
272: ||||    F                    wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
273: |||V---                  enddo
274: ||U----               enddo
275: |+-----           enddo
276: |           !
277: |V=====>           p(2:imax-1,2:jmax-1,2:kmax-1)= &
278: |                       wrk2(2:imax-1,2:jmax-1,2:kmax-1)
279: |           !
280: +------      enddo
```

演習問題３

⑥ ftraceコマンドによる結果を確認し,チューニングによる効果を確認してください.

演算性能依存	EXCLUSIVE TIME	Required B/F	Required Store B/F	Actual V. Load B/F	Actual B/F
修正前	8.75	0.39	0.07	0.26	0.33
修正後	6.89	0.50	0.08	0.33	0.41

Memory BW依存	EXCLUSIVE TIME	Required B/F	Required Store B/F	Actual V. Load B/F	Actual B/F
修正前	17.46	4.25	0.25	2.75	3.00
修正後	14.99	3.48	0.25	2.25	2.50

LLC BW依存	EXCLUSIVE TIME	Required B/F	Required Store B/F	Actual V. Load B/F	Actual B/F
修正前	11.70	4.26	0.26	1.78	2.04
修正後	10.16	3.46	0.27	1.78	2.05

4. チューニング事例紹介

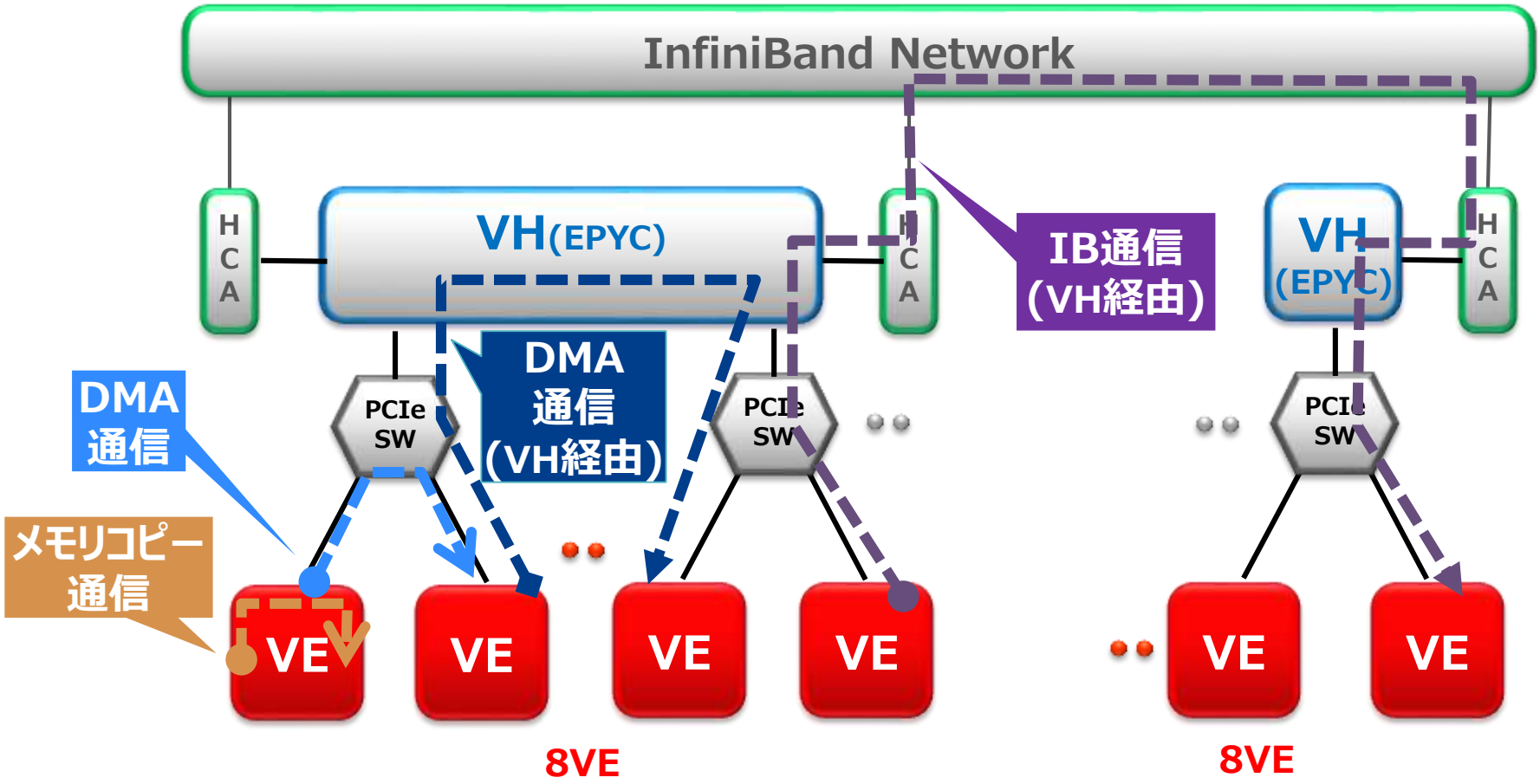
これまでチューニングを実施した事例は以下で紹介している

- 高速化推進研究活動報告(https://www.ss.cc.tohoku.ac.jp/tuning-support/)

5. MPI

MPIの通信経路について

プロセス配置やMPI通信サイズに応じて、最適な通信手段・通信経路をNECMPIが自動選択



MPI実行時の性能情報の採取方法

- 各MPIプロセスからの出力を分離するために、NEC MPI は、シェルスクリプト mpisep.sh をディレクト リ/opt/nec/ve/bin/に用意している。
- 次のように、MPI 実行指定 { MPIexec } の前に上記のスクリプトを指定すると、MPIプロセスからの出力を、プロセスごとに異なるファイルに保存することができる。

```
$ mpirun -np 2 /opt/nec/ve/bin/mpisep.sh { MPIexec }
```

- 出力の保存先は、環境変数 NMPI_SEPSELECT を使用して、次のように指定できる。ここで、**uuu** はコミュニケーターMPI_COMM_WORLD に対応する事前定義された通信範囲の識別番号、**rrr** はその通信範囲における当該プロセスのランクである。

NMPI_SEPSELECT	動　　作
1	各 MPI プロセスの標準出力だけを stdout. **uuu**:**rrr** へ保存する
2	(既定値) 各 MPI プロセスの標準エラー出力だけを stderr. **uuu**:**rrr** へ保存する
3	各 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 stdout.**uuu**:**rrr** および stderr.**uuu**:**rrr** に保存する
4	各 MPI プロセスの標準出力 および 標準エラー出力を，同一のファイル std.**uuu**:**rrr** に保存する

プログラム情報(MPI実行時)(1/3)

- MPIプログラム実行時のプログラム情報は環境変数NMPI_PROGINFを使用して採取

集約形式	Global Data部　：全プロセスの性能値の最大・最小・平均を表示 Overall Data部　：MPIプロセス全体の性能を表示 VE Card Data部：VEカード毎に集計された性能値の最大・最小・平均を表示
拡張形式	集約形式の内容の後に各MPIプロセスの性能情報が，コミュニケーターMPI_COMM_WORLDにおけるランクの昇順で表示
詳細集約形式	Global Data部　：全プロセスの詳細な性能値の最大・最小・平均を表示 Overall Data部　：MPIプロセス全体の性能を表示 VE Card Data部：VEカード毎に集計された性能値の最大・最小・平均を表示
詳細拡張形式	詳細集約形式の内容の後に各MPIプロセスの性能情報が，コミュニケーターMPI_COMM_WORLDにおけるランクの昇順で表示

- プログラム情報の表示形式は、環境変数NMPI_PROGINFを以下のように実行時に設定

NMPI_PROGINF	表示される情報
NO	性能情報を出力しない（既定値）
YES	性能情報を集約形式で出力
ALL	性能情報を拡張形式で出力
DETAIL	性能情報を詳細集約形式で出力
ALL_DETAIL	性能情報を詳細拡張形式で表示

プログラム情報(MPI実行時)(2/3)

	Global Data of 64 Vector processes	:	Min [U,R]	Max [U,R]	Average
a.	Real Time (sec)	:	0.872 [0,56]	0.877 [0,1]	0.874
b.	User Time (sec)	:	0.714 [0,0]	0.742 [0,13]	0.738
c.	Vector Time (sec)	:	0.075 [0,0]	0.632 [0,40]	0.202
d.	Inst. Count	:	74326161 [0,40]	514773956 [0,0]	202869650
e.	V. Inst. Count	:	2334382 [0,52]	14973048 [0,2]	4291220
f.	V. Element Count	:	149402019 [0,52]	824585628 [0,2]	264847523
g.	V. Load Element Count	:	94790762 [0,52]	347643327 [0,2]	144367345
h.	FLOP Count	:	16879375 [0,61]	262296925 [0,3]	36429805
i.	MOPS	:	469.678 [0,27]	3101.317 [0,0]	663.947
j.	MOPS (Real)	:	392.509 [0,27]	1474.209 [0,0]	537.951
k.	MFLOPS	:	23.010 [0,11]	628.607 [0,0]	54.028
l.	MFLOPS (Real)	:	19.271 [0,7]	299.231 [0,3]	41.623
m.	A. V. Length	:	53.847 [0,0]	64.045 [0,33]	63.265
n.	V. Op. Ratio (%)	:	43.750 [0,52]	90.960 [0,24]	51.866
o.	L1 Cache Miss (sec)	:	0.005 [0,24]	0.032 [0,0]	0.007
p.	CPU Port Conf. (sec)	:	0.000 [0,6]	0.000 [0,3]	0.000
q.	V. Arith. Exec. (sec)	:	0.007 [0,52]	0.045 [0,16]	0.013
r.	V. Load Exec. (sec)	:	0.040 [0,0]	0.427 [0,40]	0.149
s.	VLD LLC Hit Element Ratio (%)	:	99.496 [0,5]	99.989 [0,24]	99.928
t.	FMA Element Count	:	817524 [0,6]	77811759 [0,3]	6932346
u.	Power Throttling (sec)	:	0.000 [0,0]	0.000 [0,0]	0.000
v.	Thermal Throttling (sec)	:	0.000 [0,0]	0.000 [0,0]	0.000
w.	Memory Size Used (MB)	:	1606.750 [0,16]	1608.500 [0,1]	1606.875

プログラム情報(MPI実行時)(3/3)

	項目	単位	説　明
a	Real Time	秒	経過時間
b	User Time	秒	ユーザ時間
c	Vector Time	秒	ベクトル命令実行時間
d	Inst. Count		全命令実行数
e	V. Inst. Count		ベクトル命令実行数
f	V. Element Count		ベクトル命令実行要素数
g	V. Load Element Count		ベクトル命令ロード要素数
h	FLOP Count		浮動小数点データ実行数
i	MOPS		ユーザ時間1秒あたりに実行された演算数(100万単位)
j	MOPS (Real)		経過時間1秒あたりに実行された演算数(100万単位)
k	MFLOPS		ユーザ時間1秒あたりに処理された浮動小数点データ実行要素数(100万単位)
l	MFLOPS (Real)		経過時間1秒あたりに処理された浮動小数点データ実行要素数(100万単位)
m	A. V. Length		平均ベクトル長
n	V. Op. Ratio	%	ベクトル演算率／ベクトル命令を使用して演算が行われた割合
o	L1 Cache Miss	秒	L1キャッシュミス時間
p	CPU Port Conf.	秒	CPUポート競合時間(※1)
q	V. Arith Exec.	秒	ベクトル命令実行時間(※1)
r	V. Load Exec.	秒	ベクトルロード実行時間(※1)
s	VLD LLC Hit Element Ratio	%	ベクトル命令によりロードされた要素のうち、LLCからロードされた要素の比率
t	FMA Element Count		FMA命令実行要素数(※1)
u	Power Throttling	秒	電力要因によるHW停止時間(※1)
v	Thermal Throttling	秒	温度要因によるHW停止時間(※1)
w	Memory Size Used	MByte	メモリ最大使用量

MPI通信情報(MPI実行時)(1/3)

- 全MPI手続き実行所要時間，MPI通信待ち合わせ時間，送受信データ総量，および主要MPI手続き呼び出し回数を表示
- コンパイル時にオプション-mpiprof、-mpitrace、-mpiverifyまたは-ftraceを指定してMPIプログラムを作成することで利用可能
- MPI 通信情報の有無および表示形式は，環境変数NMPI_COMMINFで設定

NMPI_COMMINF	形式	表示内容
NO	MPI通信情報を表示しない	（既定値）
YES	MPI通信情報を集約形式で表示	全 MPI プロセスの MPI 通信情報の最大値，最小値，および 平均値を表示
ALL	MPI通信情報を拡張形式で表示	集約形式の内容の後に，各 MPIプロセスのMPI 通信情報が，コミュニケーターMPI_COMM_WORLD におけるランクの昇順で表示

- 環境変数NMPI_COMMINF_VIEWを追加で指定することで集約形式部分を変更

NMPI_COMMINF_VIEW	表示形式
VERTICAL	ベクトルプロセスとスカラプロセスを分けて集計し、縦に並べて表示（既定値）
HORIZONTAL	ベクトルプロセスとスカラプロセスを分けて集計し、横に並べて表示
MERGED	ベクトルプロセスとスカラプロセスをまとめて集計し表示。ベクト ルプロセスにのみ対応した項目は行末に(V)が付く

MPI通信情報(MPI実行時)(2/3)

- 出力例
  NMPI_COMMINF=YES　指定時

MPI Communication Information of 64 Vector processes

		Min [U,R]	Max [U,R]	Average
1	Real MPI Idle Time (sec) :	0.004 [0,0]	0.633 [0,40]	0.388
2	User MPI Idle Time (sec) :	0.004 [0,0]	0.633 [0,40]	0.388
3	Total real MPI Time (sec) :	0.147 [0,0]	0.774 [0,11]	0.749
4	Send count :	0 [0,0]	1 [0,1]	0
5	Memory Transfer :	0 [0,0]	1 [0,1]	0
6	DMA Transfer :	0 [0,0]	0 [0,0]	0
7	Recv count :	0 [0,1]	5 [0,0]	0
8	Memory Transfer :	0 [0,1]	5 [0,0]	0
9	DMA Transfer :	0 [0,0]	0 [0,0]	0
10	Barrier count :	15 [0,0]	15 [0,0]	15
	Bcast count :	858 [0,0]	858 [0,0]	858
	Reduce count :	5 [0,0]	5 [0,0]	5
	Allreduce count :	163 [0,6]	282 [0,0]	172
	Scan count :	0 [0,0]	0 [0,0]	0
	Exscan count :	0 [0,0]	0 [0,0]	0
	Redscat count :	0 [0,0]	0 [0,0]	0
	Redscat_block count :	0 [0,0]	0 [0,0]	0
	Gather count :	3 [0,0]	3 [0,0]	3
	Gatherv count :	0 [0,0]	0 [0,0]	0
	Allgather count :	0 [0,0]	0 [0,0]	0
	Allgatherv count :	0 [0,0]	0 [0,0]	0
	Scatter count :	0 [0,0]	0 [0,0]	0
	Scatterv count :	0 [0,0]	0 [0,0]	0
	Alltoall count :	0 [0,0]	0 [0,0]	0
	Alltoallv count :	0 [0,0]	0 [0,0]	0
	Alltoallw count :	0 [0,0]	0 [0,0]	0
11	Neighbor Allgather count :	0 [0,0]	0 [0,0]	0
	Neighbor Allgatherv count :	0 [0,0]	0 [0,0]	0
	Neighbor Alltoall count :	0 [0,0]	0 [0,0]	0
	Neighbor Alltoallv count :	0 [0,0]	0 [0,0]	0
	Neighbor Alltoallw count :	0 [0,0]	0 [0,0]	0
12	Number of bytes sent :	0 [0,0]	8 [0,1]	1
13	Memory Transfer :	0 [0,0]	8 [0,1]	1
14	DMA Transfer :	0 [0,0]	0 [0,0]	0
15	Number of bytes recvd :	0 [0,1]	40 [0,0]	1
16	Memory Transfer :	0 [0,1]	40 [0,0]	1
17	DMA Transfer :	0 [0,0]	0 [0,0]	0
18	Put count :	0 [0,0]	0 [0,0]	0
19	Get count :	0 [0,0]	0 [0,0]	0
20	Accumulate count :	0 [0,0]	0 [0,0]	0
21	Number of bytes put :	0 [0,0]	0 [0,0]	0
22	Number of bytes got :	0 [0,0]	0 [0,0]	0
23	Number of bytes accum :	0 [0,0]	0 [0,0]	0

MPI通信情報(MPI実行時)(3/3)

	項目	単位	説　明
1	Real MPI Idle Time	秒	メッセージ待ちに費やした経過時間
2	User MPI Idle Time	秒	メッセージ待ちに費やしたユーザCPU時間
3	Total real MPI TIME	秒	MPI手続きの実行に費やした経過時間
4	Send count		1対1送信手続きの呼び出し回数
5	Memory Transfer		メモリコピーを使用する1対1送信手続きの呼び出し回数
6	DMA Transfer		DMA転送を使用する1対1送信手続きの呼び出し回数
7	Recv count		1対1受信手続きの呼び出し回数
8	Memory Transfer		メモリコピーを使用する1対1受信手続きの呼び出し回数
9	DMA Transfer		DMA転送を使用する1対1受信手続きの呼び出し回数
10	Barrier count		手続 MPI_BARRIER および MPI_IBARRIER の呼出し回数
11	Neighbor Allgather count		手続 MPI_NEIGHBOR_ALLGATHER および MPI_INEIGHBOR_ALLGATHER の呼出し回数
12	Number of bytes sent	Byte	1 対 1 送信手続により送信したバイト数
13	Memory Transfer	Byte	メモリコピーを使用する 1 対 1 送信手続により送信したバイト 数
14	DMA Transfer	Byte	DMA 転送を使用する 1 対 1 送信手続により送信したバイト数
15	Number of bytes recvd	Byte	1 対 1 送信手続により受信したバイト数
16	Memory Transfer	Byte	メモリコピーを使用する 1 対 1 受信手続により送信したバイト 数
17	DMA Transfer	Byte	DMA 転送を使用する 1 対 1 受信手続により送信したバイト数
18	Put count		手続 MPI_PUTおよびMPI_RPUTの呼出し回数
19	Get count		手続 MPI_GETおよびMPI_RGETの呼出し回数
20	Accumulate count		手続MPI_ACCUMULATE、MPI_RACCUMULATE、 MPI_GET_ACCUMULATE、MPI_RGET_ACCUMULATE、 MPI_FETCH_AND_OP およびMPI_COMPARE_AND_SWAP の呼出し回数
21	Number of bytes put	Byte	手続 MPI_PUT および MPI_RPUT により送信したバイト数
22	Number of bytes got	Byte	手続 MPI_GET および MPI_RGET により受信したバイト数
23	Number of bytes accum	Byte	手続 MPI_ACCUMULATE、MPI_RACCUMULATE、 MPI_GET_ACCUMULATE、MPI_RGET_ACCUMULATE、 MPI_FETCH_AND_OP および MPI_COMPARE_AND_SWAP により積算したバイト数

簡易性能解析(ftrace)(MPI実行時)(1/3)

- MPI実行時はプロセスごとにftrace.outファイル(ftrace.out.X.X)が作成される
- ftraceコマンドで複数のftrace.outファイルを指定可能

(例)4プロセス実行時には,ftrace.out.0.0～ftrace.out.0.3が作成

①プロセス0番の情報のみ出力したい場合
　% ftrace –f ftrace.out.0.0
②全プロセスの情報を出力したい場合(コストの上位10ルーチンを表示)
　% ftrace –f ftrace.out.*
③全プロセスの全ルーチンの情報を出力したい場合
　% ftrace –f ftrace.out.* -all

簡易性能解析(ftrace)(MPI実行時)(2/3)

```
 ①          ②          ③          ④          ⑤          ⑥          ⑦     ⑧        ⑨
 ELAPSED     COMM.TIME  COMM.TIME   IDLE TIME  IDLE TIME  AVER.LEN      COUNT  TOTAL LEN PROC.NAME
 TIME[sec]       [sec]  / ELAPSED        [sec]  / ELAPSED    [byte]              [byte]

    12.444       0.000                 0.000                 0.0          0       0.0  funcA
     9.420       0.000                 0.000                 0.0          0       0.0  funcB
     7.946       0.000                 0.000                 0.0          0       0.0  funcG
     7.688       0.000                 0.000                 0.0          0       0.0  funcC
     7.372       0.000                 0.000                 0.0          0       0.0  funcH
     5.897       0.000                 0.000                 0.0          0       0.0  funcD
     5.653       0.000                 0.000                 0.0          0       0.0  funcE
     1.699       1.475                 0.756                 3.1K    642560       1.9G funcF
     1.073       1.054                 0.987                 1.0M      4064       4.0G funcI
     0.704       0.045                 0.045                80.0         4     320.0  funcK
 ----------------------------------------------------------------------------------------------------

FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS  V.OP  AVER.    VECTOR L1CACHE CPU PORT VLD LLC PROC.NAME
           TIME[sec](  % )    [msec]                    RATIO V.LEN      TIME     MISS     CONF HIT E.%

     1012    49.093( 24.0)    48.511  23317.2  14001.4  96.97  83.2    42.132   5.511    0.000   80.32 funcA
      253    12.089           47.784  23666.9  14215.9  97.00  83.2    10.431   1.352    0.000   79.40  0.0
      253    12.442           49.177  23009.2  13811.8  96.93  83.2    10.617   1.406    0.000   81.26  0.1
      253    12.118           47.899  23607.4  14180.5  97.00  83.2    10.463   1.349    0.000   79.36  0.2
      253    12.444           49.185  23002.8  13808.2  96.93  83.2    10.622   1.404    0.000   81.26  0.3
...
----------------------------------------------------------------------------------------------------
 54851346   204.569(100.0)     0.004  22508.5  12210.7  95.64  76.5   154.524  17.740   13.916   90.29 total

   ①          ②          ③          ④          ⑤          ⑥          ⑦     ⑧        ⑨
   ELAPSED     COMM.TIME  COMM.TIME   IDLE TIME  IDLE TIME  AVER.LEN      COUNT  TOTAL LEN PROC.NAME
    TIME[sec]       [sec]  / ELAPSED        [sec]  / ELAPSED    [byte]              [byte]

      12.444       0.000                 0.000                 0.0          0       0.0  funcA
      12.090       0.000       0.000     0.000       0.000     0.0          0       0.0   0.0
      12.442       0.000       0.000     0.000       0.000     0.0          0       0.0   0.1
      12.119       0.000       0.000     0.000       0.000     0.0          0       0.0   0.2
      12.444       0.000       0.000     0.000       0.000     0.0          0       0.0   0.3
...
```

各プロセスの値

簡易性能解析(ftrace)(MPI実行時)(3/3)

	項目	単位	説　明
①	ELAPSED TIME	秒	経過時間
②	COMM.TIME	秒	MPI 通信の送受信に要した経過時間
③	COMM.TIME / ELAPSED		関数内での経過時間に対する MPI COMM.TIME の比率
④	IDLE TIME	秒	MPI 通信の通信を行うまでの待ち時間、および 同期待ちに要した経過時間
⑤	IDLE TIME / ELAPSED		関数内での経過時間に対する MPI IDLE.TIME の比率
⑥	AVER.LEN	Byte	MPI 通信 1 回当たりの平均通信量(TOTAL LEN / COUNT)。バイト、キロバイト、メガバイト、 ギガバイト、テラバイト、ペタバイト単位で表示
⑦	COUNT		MPI 通信回数
⑧	TOTAL LEN	Byte	MPI 通信の通信量。バイト、キロバイト、メガ、バイト、ギガバイト、テラバイト、ペタバイト単 位で表示
⑨	PROC.NAME		関数名、または MPI プロセス (MPIUNIVERSE.MPIRANK)

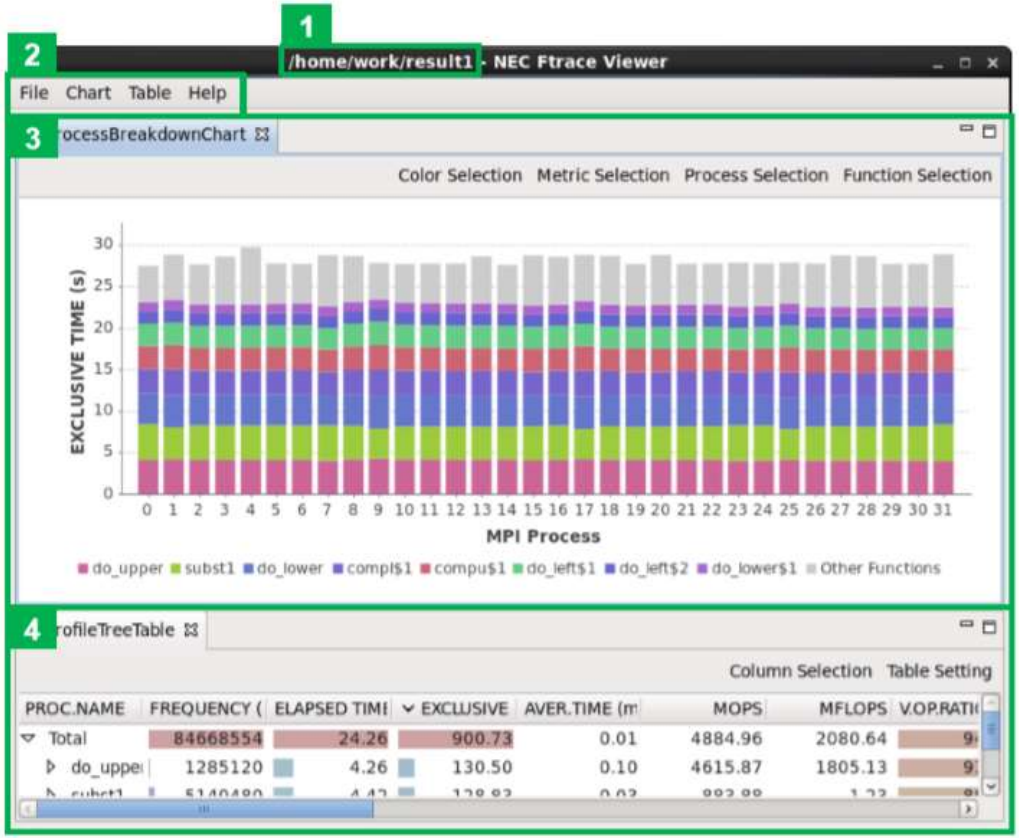
NEC Ftrace Viewer (1/10)

- NEC Ftrace Viewer は、Fortran/C/C++プログラムのチューニングを支援するGUI ツール
  - FTRACEによる性能測定結果をグラフィカルに表示
  - 自動並列化機能、OpenMP、MPI を利用したプログラムに対応し、スレッド・MPI プロセス ごとの実行時間、MPI プロセス間の通信時間を表示でき、性能ボトルネックやそれらのロー ドインバランスを容易に把握可能
  - 性能値に基づいた関数・スレッド・MPI プロセスの絞り込み表示や多彩なグラフ形式により、 シングルスレッドプログラムから数千プロセスに及ぶ MPI プログラムまで、様々なプログラ ムの性能解析をサポート

  - NEC Ftrace ViewerはX Windows Systemを利用しているため、ユーザ側でXサーバを起動する必要がある。
  - 詳細は「NEC Ftrace Viewer ユーザガイド」を参照

NEC Ftrace Viewer (2/10)

$ /opt/nec/ve/ftraceviewer/ftraceviewer
または
$ /opt/nec/ve/ftraceviewer/ftraceviewer –f ftrace.out.*



① ウィンドウタイトル
② メニュー
③ グラフ表示領域
④ テーブル表示領域

NEC Ftrace Viewer (3/10)

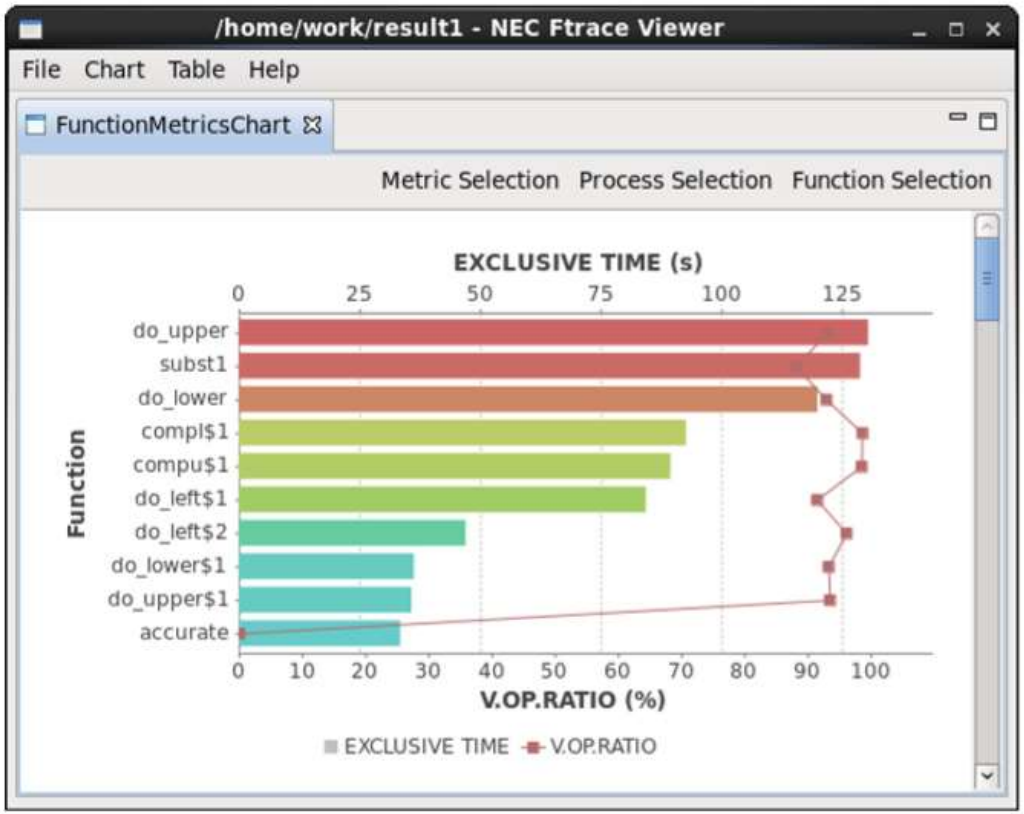
- 表示するグラフの種類は以下のとおり

グラフ名	内　容
Function Metric Chart	実行コードごとに性能値を表示するグラフ 実行時間の長い関数を発見するために利用 (※Function Breakdown Chartは自動並列・OpenMPプログラムのみ)
Function Breakdown Chart	
Function Statistics Chart	
Process Metric Chart	実行コンテキストごとに性能値を表示するグラフMPI プロセス間やスレッド間のロードインバランスを発見するために利用 (※MPI Communication ChartはMPIプログラムのみ)
Process Breakdown Chart	
MPI Communication Chart	
Process Histogram	実行コンテキストの性能値分布を表示するグラフ 大量のMPI プロセスから性能の悪いMPI プロセスを発見するために利用

※グラフでは、特に指定がない限り、実行コードを「関数(Function)」、
実行コンテキストを「プロセ ス(Process)」とも呼ぶ

NEC Ftrace Viewer (4/10)

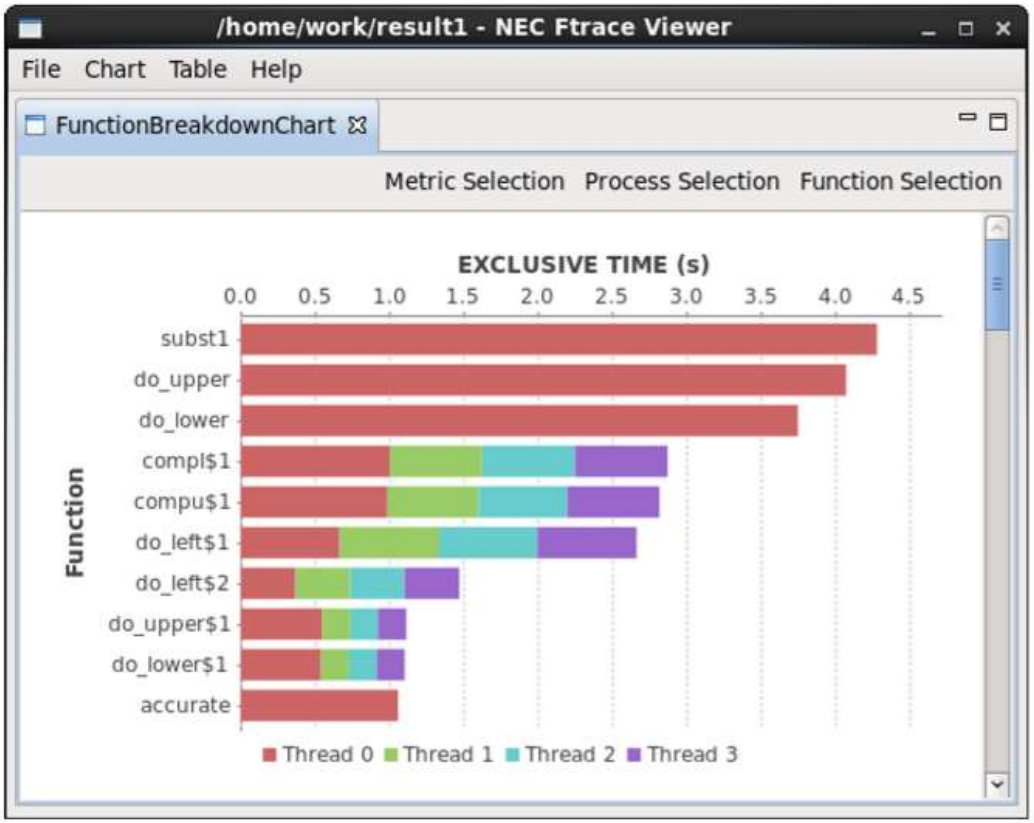
- **Function Metric Chart**…関数ごとに、性能値を色つきの棒グラフと折れ線グラフで表示。棒グラフの軸は上に、折れ 線グラフの軸は下に表示される。折れ線グラフで表示する性能項目は任意の種類選択可能。



- 棒グラフの色は選択した関数の中で、最大値を赤、中間を緑、最小値を青でグラデーション表示

NEC Ftrace Viewer (5/10)

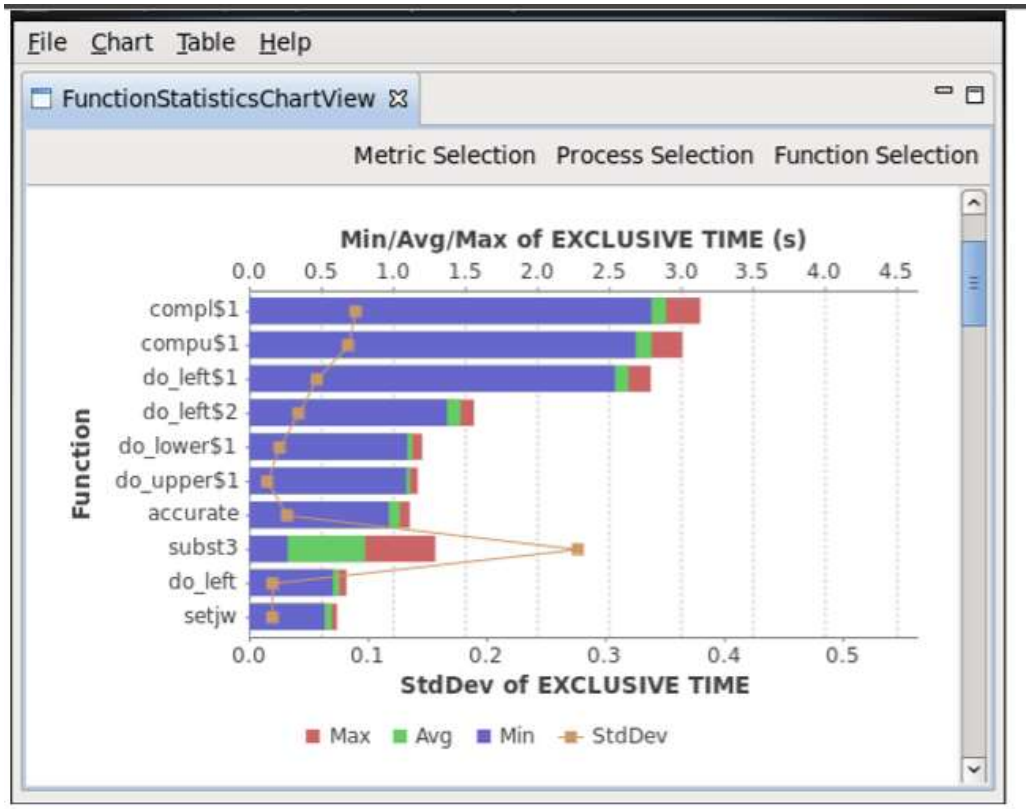
- **Function Breakdown Chart** …関数ごとに、各スレッドの性能値を色分けした棒グラフで表示。例えば、下記の例では、1 スレッドで実行されている関数subst1は1色で表示され、4 スレッドで実行されている関数compl$1はスレッドごとに4色に色分けされて表示される。

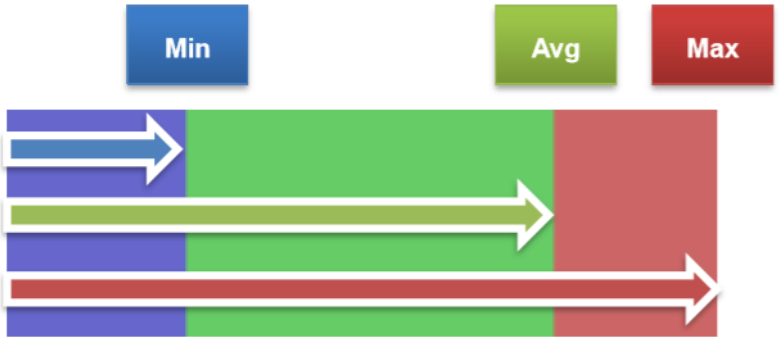


- Function Metrics Chart と同様に、折れ線グラフで複数の性能値を同時に表示

NEC Ftrace Viewer (6/10)

- **Function Statistics Chart**…関数ごとに、プロセスの性能値の最大・最小・平均・標準偏差を表示

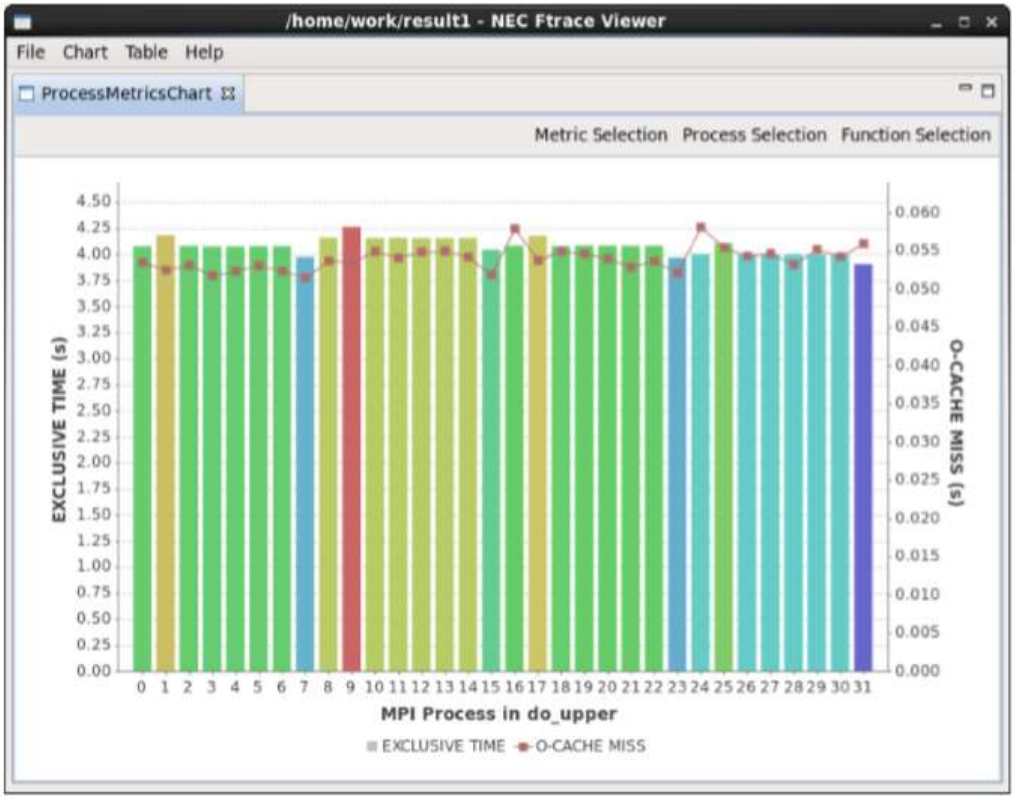




- 棒グラフは右記の様に、左端を基準にして、青から緑に変わる箇所が最小値、緑から赤に変わる箇 所が平均値、赤色の右端が最大値を表している。

NEC Ftrace Viewer (7/10)

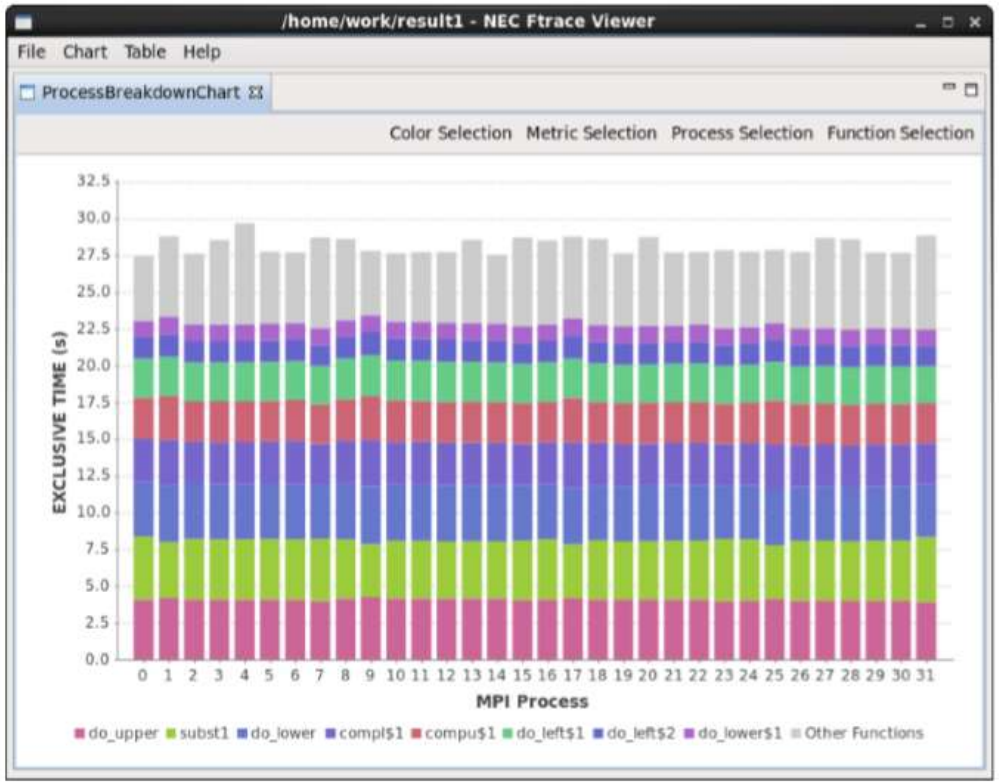
- **Process Metrics Chart**…指定した関数に対して、プロセスごとの性能値を色つきの棒グラフと折れ線グラフで表示。棒グラフの軸は左に、折れ線グラフの軸は右に表示。折れ線グラフで表示する性能項目は任意の種類が選択可能



- 棒グラフの色は選択したプロセスの中で、最大値を赤、中間を緑、最小値を青でグラデーション表示。上記の例では、ランク番号が9のプロセスが最大値、ランク番号が31のプロセスが最小値となる

NEC Ftrace Viewer (8/10)

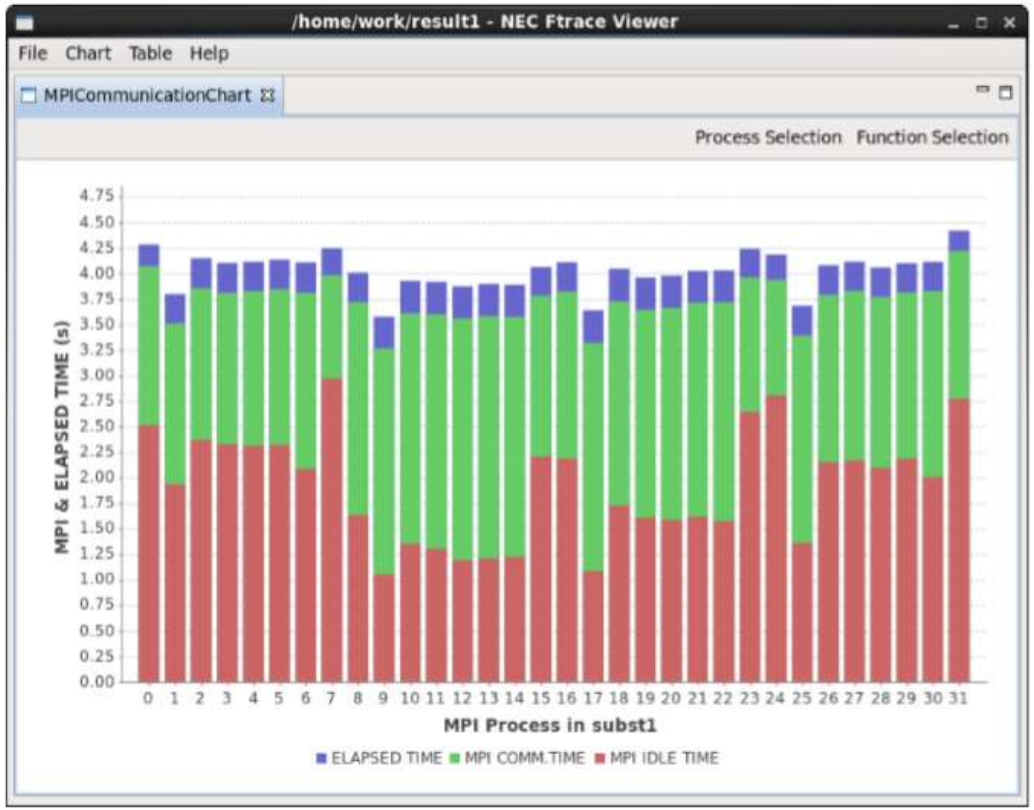
- **Process Breakdown Chart**…MPI プロセスごとに性能値の大きな関数を下から順に積み上げて表示。性能値が小さな関数はOtherとして灰色でまとめられる。



- 個別に積み上げる関数の指定と積み上げ順の変更は[Function Selection]で行う
- チェックが入った関数が上から順番に選択され、個別にグラフに積み上げられる
- チェックが入っていない関数 は Other Functions として一つにまとめられる

NEC Ftrace Viewer (9/10)

- **MPI Communication Chart**…指定した関数に対して、MPI プロセスごとの経過時間・MPI 通信時間・MPI 通信待ち時間を表示

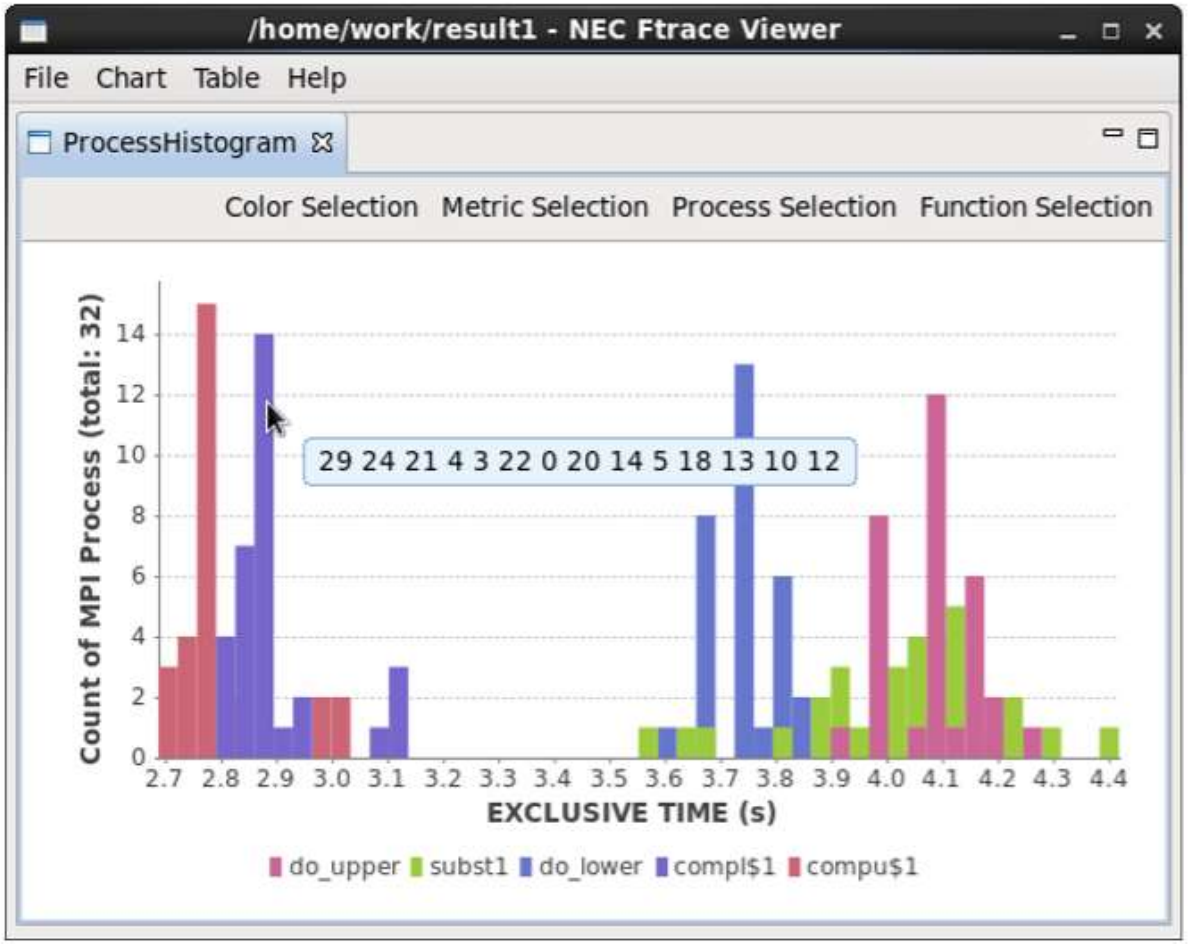


※MPI通信時間はMPI通信待ち時間を含むため、待ち時間を含まないMPI通信処理時間は、緑色の範囲になる。同様に、経過時間はMPI通信時間を含むため、ユーザの計算コードの実行など MPI通信以外の処理にかかった時間は青色の範囲になる

- 棒グラフは右記の様に、下端を基準にして、赤から緑への変わり目がMPI通信待ち時間(MPI IDLE TIME)、緑から青への変わり目がMPI通信時間(MPI COMM.TIME)、青色の上端が経過時間(ELAPSED TIME)を表している

NEC Ftrace Viewer (10/10)

- **Process Histogram**…MPIプロセスの性能分布をヒストグラムで表示。横軸は性能値、縦軸はその性能値を取るMPIプロセスの個数を表している。ヒストグラムにフォーカスを合わせるとMPIランク番号がポップアップする



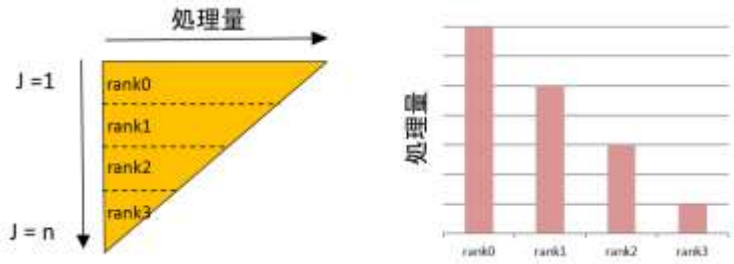
MPIプログラムの分析 (1/3)

- 並列プログラムの性能を分析する際に，転送時間がコストの上位を占める場合は，データ転送処理のコストであるか，演算のインバランスを吸収している結果であるか見極める必要がある.

```
31: +------>          do it=1,m
32: |                  sum1=0.0d0
33: |                  call sub1(it)
34: |                  call sub2
35: |                  sum3=sum3+sum2
36: +------           enddo

45:                   subroutine sub1(it)
46:                   use sample
47: +------>          do j=ist,ied
48: |V----->           do i=1,j
49: ||      A            sum1 = sum1 + p(it)*a(i,j) - b(i,j)*c(i,j)
50: |V-----             enddo
51: +------           enddo
52:                   return
53:                   end
54:                   subroutine sub2
55:                   use mpi
56:                   use sample
57:                   call MPI_REDUCE(sum1,sum2,1,MPI_REAL8,MPI_SUM,0,
58:                  +                MPI_COMM_WORLD,ierr)
59:                   return
60:                   end
```

- 左記のサンプルプログラムは，sub1で演算を行っており，sub2で各プロセスの結果をMPI_ALLREDUCEのリダクション処理(総和)を行っている
- sub1の演算量は下記に示すように，プロセスに均等ではない.



MPIプログラムの分析 (2/3)

- サンプルプログラムをSX-ACE 1ノード(4コア)で実行した結果は以下．

PROC.NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	ADB HIT ELEM.%
sub1	8000	72.042(57.0)	9.005	26727.3	15022.2	99.34	251.1	72.038	0.001	0.003	0.000	48.303	0.06
0.0	2000	4.283	2.142	29622.2	16162.2	99.20	238.2	4.282	0.000	0.001	0.000	2.437	0.11
0.1	2000	16.044	8.022	22638.8	12680.9	99.32	249.5	16.043	0.000	0.001	0.000	11.365	0.07
0.2	2000	22.928	11.464	26148.1	14727.0	99.35	252.0	22.928	0.000	0.001	0.000	15.581	0.05
0.3	2000	28.786	14.393	29036.5	16392.6	99.36	253.1	28.786	0.000	0.001	0.000	18.920	0.06
sub2	8000	42.859(33.9)	5.357	141.0	0.0	10.85	13.0	11.551	0.020	0.024	0.000	7.695	0.00
0.0	2000	24.218	12.109	140.4	0.0	10.90	13.0	7.015	0.014	0.018	0.000	4.852	0.00
0.1	2000	12.770	6.385	140.4	0.0	10.86	13.0	3.136	0.002	0.002	0.000	1.974	0.00
0.2	2000	5.865	2.933	144.2	0.0	10.64	13.0	1.397	0.003	0.002	0.000	0.869	0.00
0.3	2000	0.006	0.003	291.1	1.1	2.57	3.0	0.002	0.001	0.001	0.000	0.000	0.00

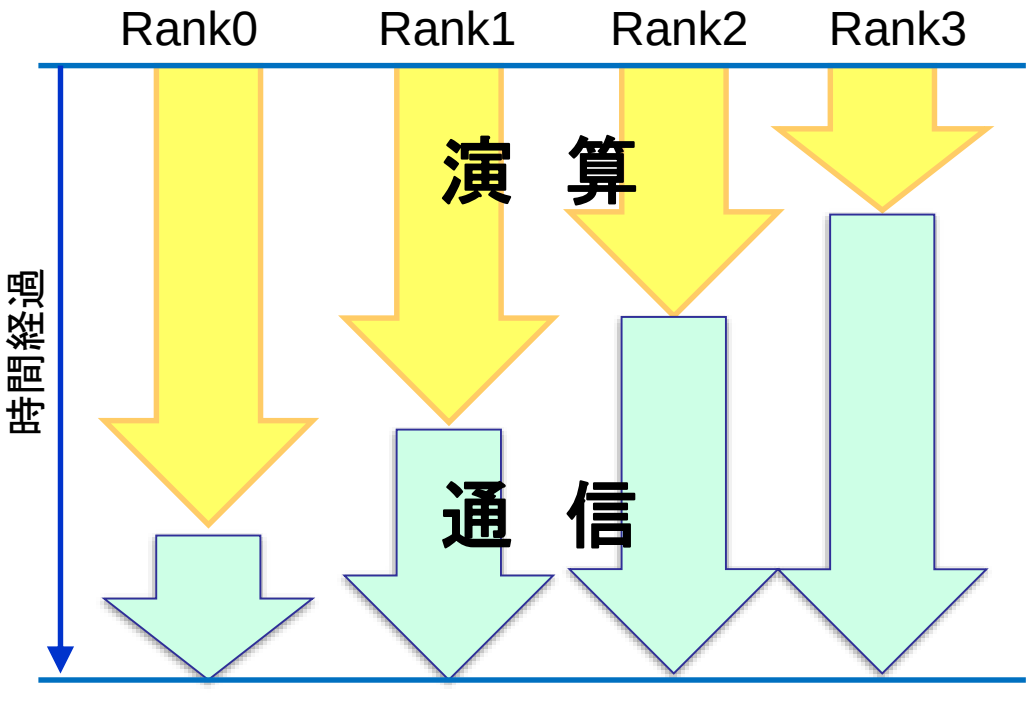
- 演算を行っているsub1のコスト(57%)に対して，MPI_ALLREDUCEを行っているsub2のコストが34%と比較的大きい．これはプロセスの待ち時間がMPI_ALLREDUCEにカウントされるためである．

PROC.NAME	ELAPSED TIME[sec]	COMM.TIME [sec]	COMM.TIME / ELAPSED	IDLE TIME [sec]	IDLE TIME / ELAPSED	AVER.LEN [byte]	COUNT	TOTAL LEN [byte]
sub2	24.502	24.499		5.670		10.0	8000	78.1K
0.0	24.502	24.499	1.000	5.670	0.231	16.0	2000	31.2K
0.1	12.771	12.770	1.000	2.462	0.193	8.0	2000	15.6K
0.2	5.869	5.867	1.000	1.151	0.196	8.0	2000	15.6K
0.3	0.006	0.005	0.775	0.000	0.000	8.0	2000	15.6K

※転送時間にインバランスがある

MPIプログラムの分析 (3/3)

- プログラムのステップ当たりの時間経過は次の通りである.



※集団通信は全プロセスの同期を取って通信を実行する為，演算終了の待ち合わせを行う.

- sub2をcallする前にMPI_BARRIERで同期を取ることで，sub2にプロセスの待ち合わせ時間を含めない.

PROC.NAME	ELAPSED TIME[sec]	COMM.TIME [sec]	COMM.TIME / ELAPSED	IDLE TIME [sec]	IDLE TIME / ELAPSED	AVER.LEN [byte]	COUNT	TOTAL LEN [byte]
sub2	0.020	0.018		0.004		10.0	8000	78.1K
0.0	0.020	0.018	0.925	0.003	0.155	16.0	2000	31.2K
0.1	0.017	0.016	0.927	0.004	0.209	8.0	2000	15.6K
0.2	0.006	0.005	0.791	0.000	0.008	8.0	2000	15.6K
0.3	0.011	0.009	0.812	0.000	0.022	8.0	2000	15.6K