大阪大学サイバーメディアセンター
Cybermedia Center Osaka University

Empowered by Innovation NEC

SX-ACE 並列プログラミング入門 (HPF)

大阪大学サイバーメディアセンター
日本電気株式会社

目次

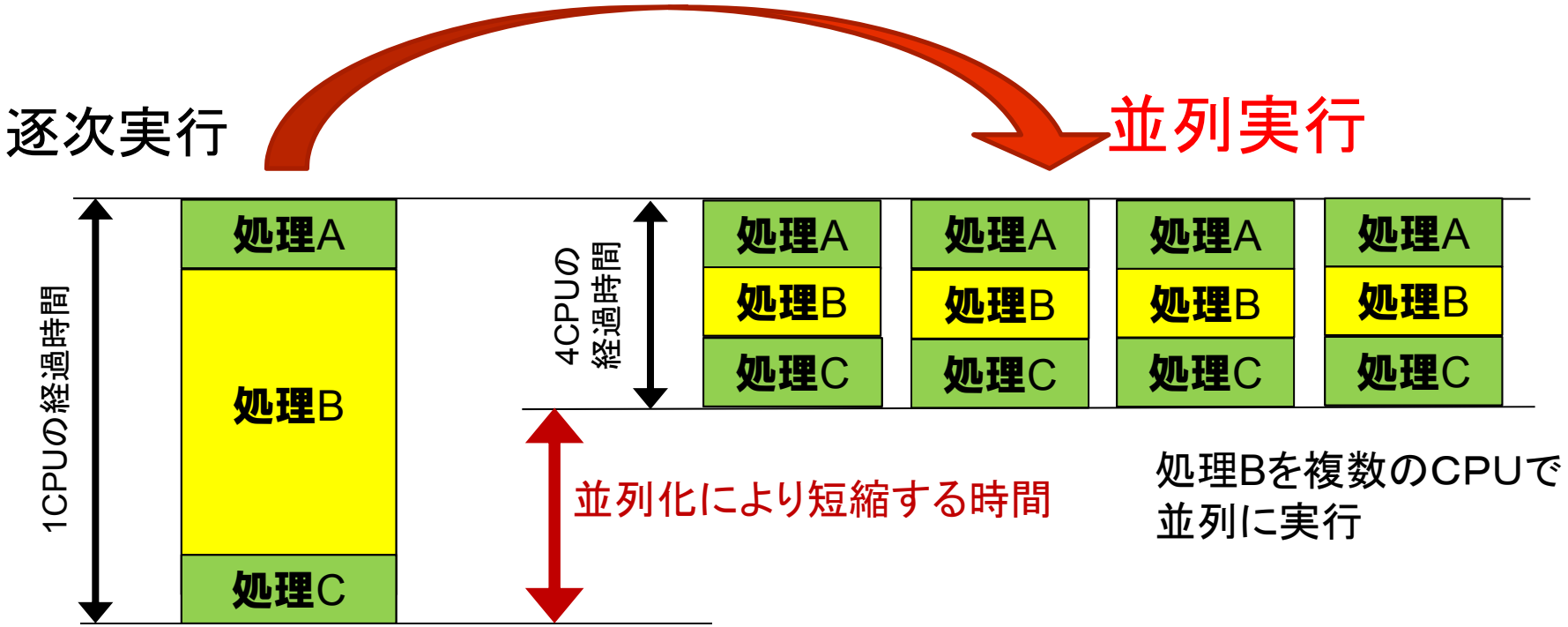
1. HPF概要

並列処理・並列実行

- 仕事（処理）を複数のコアまたはCPUに分割し、同時に実行すること

並列化

- 並列処理を可能とするために、処理の分割を行うこと



並列化の効果

行列積プログラム

```
      implicit real(8)(a-h,o-z)
      parameter ( n=15360 )
      real(8) a(n,n),b(n,n),c(n,n)
      real(4) etime,cp1(2),cp2(2),t1,t2,t3
      do j = 1,n
        do i = 1,n
          a(i,j) = 0.0d0
          b(i,j) = n+1-max(i,j)
          c(i,j) = n+1-max(i,j)
        enddo
      enddo
      write(6,50) ' Matrix Size = ',n
   50 format(1x,a,i5)
      t1=etime(cp1)
      do j=1,n
        do k=1,n
          do i=1,n
            a(i,j)=a(i,j)+b(i,k)*c(k,j)
          end do
        end do
      end do
      t2=etime(cp2)
      t3=cp2(1)-cp1(1)
      write(6,60) ' Execution Time = ',t2,' sec',' A(n,n) = ',a(n,n)
   60 format(1x,a,f10.3,a,1x,a,d24.15)
      stop
      end
```

- SX-ACE　1coreの実行時間は約114.8秒

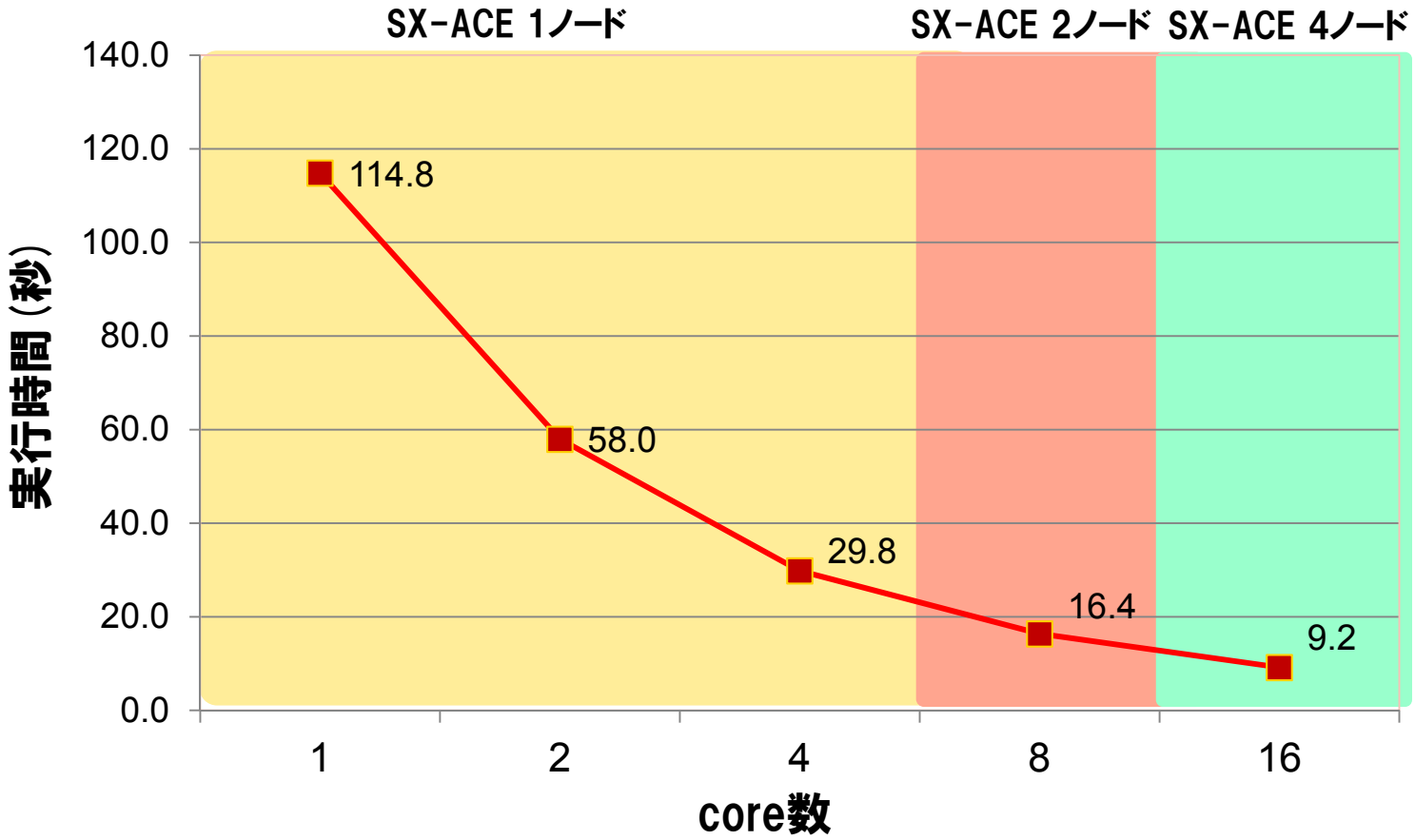
```
Matrix Size = 15360
Execution Time =    114.876 sec  A(n,n) =    0.153600000000000D+05

 ******  Program Information  ******
Real Time (sec)                 :          114.830190
User Time (sec)                 :          114.820321
Sys  Time (sec)                 :            0.005606
Vector Time (sec)               :          114.820061
Inst. Count                     :        56231275741
V. Inst. Count                  :        35849130439
V. Element Count                :      9175961813328
V. Load Element Count           :       170106224680
FLOP Count                      :      7247757312103
MOPS                            :        80093.348273
MFLOPS                          :        63122.601026
A. V. Length                    :          255.960513
V. Op. Ratio (%)                :           99.778367
Memory Size (MB)                :         5568.031250
MIPS                            :          489.732786
I-Cache (sec)                   :            0.000232
O-Cache (sec)                   :            0.000377
Bank Conflict Time
  CPU Port Conf. (sec)          :            0.000000
  Memory Network Conf. (sec)    :            1.095781
ADB Hit Element Ratio (%)       :            0.000000
```

- 複数のcoreを用いることで実行時間を短縮することが可能に

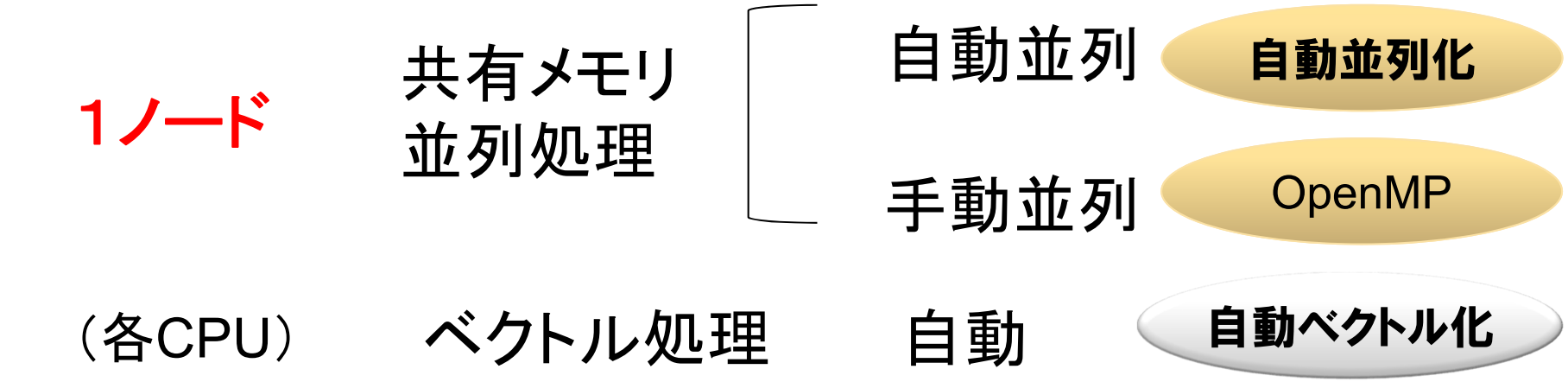
並列化の効果

- 並列化により複数のCPUを利用し，実行時間を短縮
- MPIを用いることで，SX-ACEの複数ノードが利用可能



1ノードでは29.8秒まで．さらに時間を短縮するためには複数ノード必要．

SX-ACEのプログラミング環境

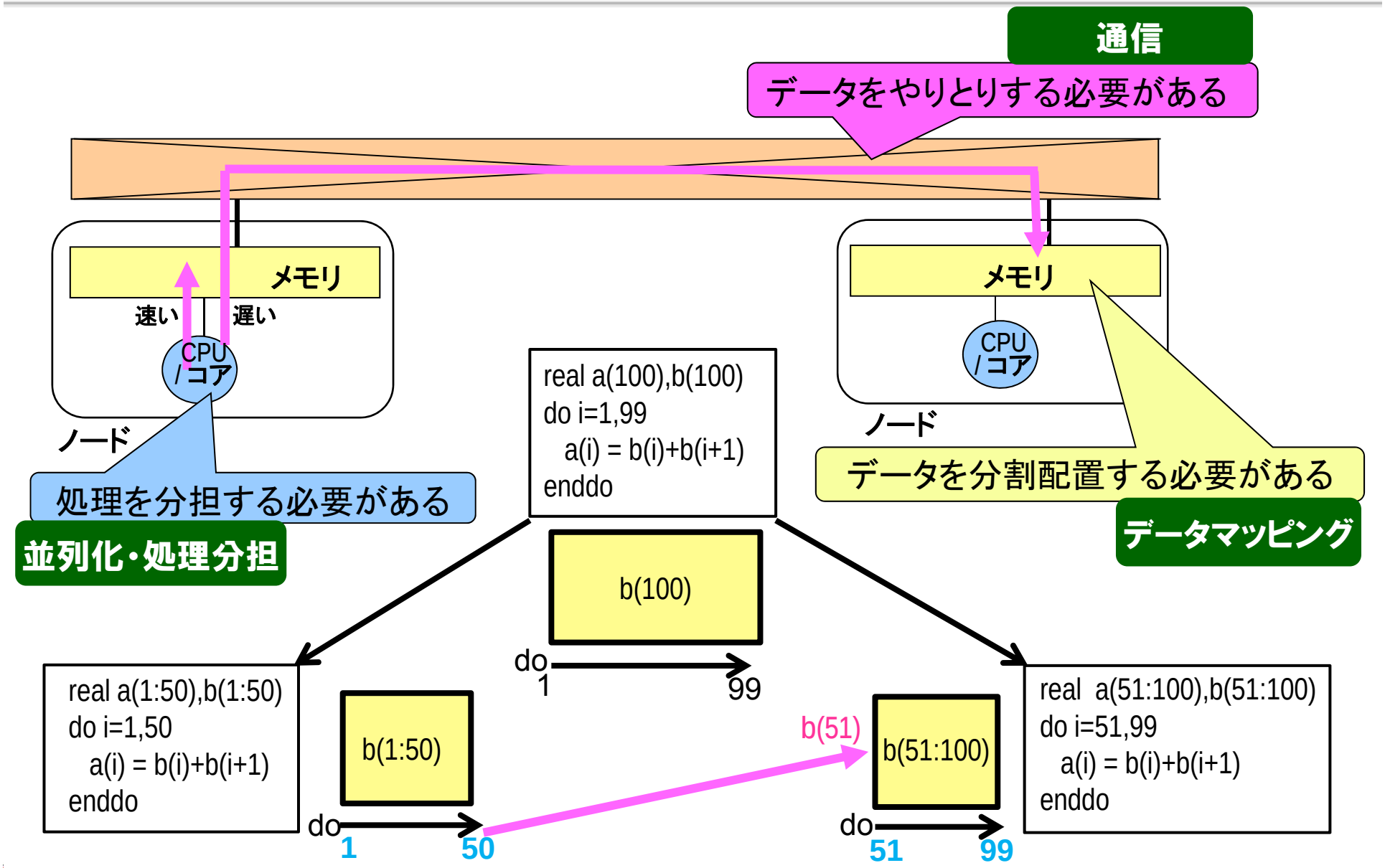


HPFとは

HPF(High Performance Fortran)とは？

- Fortranの分散メモリ型並列計算機向け拡張仕様で以下の特徴を有する
- 国際的な標準仕様(並列処理の主要ベンダ，大学，研究機関が共同で仕様策定)
- 記述が平易(Fortran＋コメント形式の指示文)
- 上級ユーザは細かな制御も可能(通信の制御，部分的にMPIを利用したチューニング等)

分散並列化のために必要な作業



HPFとMPI

HPFは，分散並列プログラム開発の3つの局面

① 複数プロセッサへのデータの分割配置(マッピング)

② 複数プロセッサへの並列化・処理の分担

③ 複数プロセッサ間の通信処理の挿入

のうち，②と③を自動化

	MPI	HPF
①データの分割配置	手動	手動
②並列化・処理の分担	手動	自動
③通信処理の挿入	手動	自動

※MPI(Message Passing Interface)

HPFとMPI

1,000行程度の流体解析コードの並列化による比較

		SX-ACE/4Core	SX-ACE/64Core
オリジナルコード	約1,000行	1,866秒	
MPI化コード	すべて手動で並列化 約580行の追加・修正必要		677秒
HPF化コード	約40行の追加のみ		694秒

上記コードの場合，少ないプログラム修正(追加)でMPI化コードと比較して遜色のない並列効果を達成することが可能.

コンパイル・実行コマンド

HPFプログラムのコンパイル

```
sxhpf  -Mlist2  [オプション]  ソースファイル名
```

※ オプションはsxf90と同様.

※ -Mlist2：並列化情報リストを採取するオプション(推奨)

HPFプログラムの実行(MPIプログラムと同じ)

```
mpirun  -nn  [ノード数]  –np  [総MPIプロセス数]  ロードモジュール名
```

または

```
mpirun  -nn  [ノード数]  –nnp  [ノード当たりのMPIプロセス数]  ロードモジュール名
```

実行スクリプト例

32mpi 4smpのジョブを32ノードで実行する際のスクリプト例

```
#!/bin/csh

#PBS -q ACE
#PBS -T mpisx
#PBS -b 32
#PBS -l cpunum_job=4,memsz_job=60GB,elapstim_req=20:00:00
#PBS -N Test_Job
#PBS -v F_RSVTASK=4

cd $PBS_O_WORKDIR

mpirun -nn 32 -nnp 1 ./a.out
```

NQSIIオプション(#PBSで指定)
-q　ジョブクラス名を指定
-T　SX向けMPI/HPFジョブであることを宣言
-b　使用ノード数を指定
-l　使用CPU数、経過時間、メモリ容量の申告
-j o 標準エラー出力を標準出力と同じファイルへ出力する
-N　ジョブ名を指定
-v (実行する全てのノードに対して)環境変数を設定する

F_RSVTASK=4を実行する全てのノードに対して設定する

$PBS_O_WORKDIR:ジョブスクリプトをqsubしたディレクトリ

ジョブクラス一覧

ジョブクラス	プロセス数	メモリ容量	利用方法
DBG	32	480GB	共有
ACE	1024	15TB	
myACE	4*占有ノード数	60GB*占有ノード数	占有

詳細は http://www.hpc.cmc.osaka-u.ac.jp/system/manual/sx-ace/jobclass_sxace/ を参照

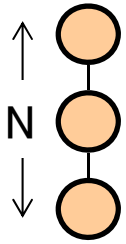
2. HPFの基本機能

プロセッサの構成を宣言する	PROCESSORS指示文
配列を分散する	DISTRIBUTE指示文
並列化可能であることを明示する	INDEPENDENT指示文 [オプション] ◆作業変数指定: NEW節 ◆集計変数指定: REDUCTION節
実行プロセッサを指定する	ON指示構文 [オプション] ◆通信不要を明示: LOCAL節

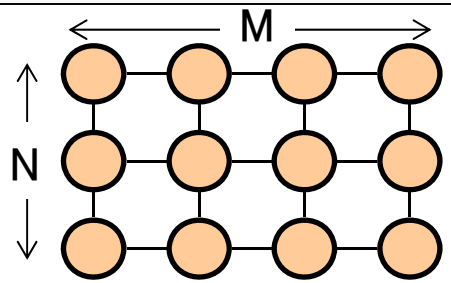
PROCESSORS指示文

プロセッサの構成を宣言する

!HPF$ PROCESSORS P(N)



!HPF$ PROCESSORS P(N,M)



書式: プロセッサ構成*pi*を宣言する(配列の宣言と同様)

PROCESSORS *p1*(上下限,...), *p2*(上下限,...), ...

特徴:

- プロセッサ構成の各要素は抽象プロセッサと呼ばれ、HPF/SX V2ではプロセスと一対一に対応する
- 組込み関数number_of_processors()を使用して、プログラムを修正せずに、プロセッサ数を実行時に決定することも可能
  - 例: !HPF$ PROCESSORS P(number_of_processors())
- 省略すると、実行時のプロセス数から、処理系が自動的に決定する

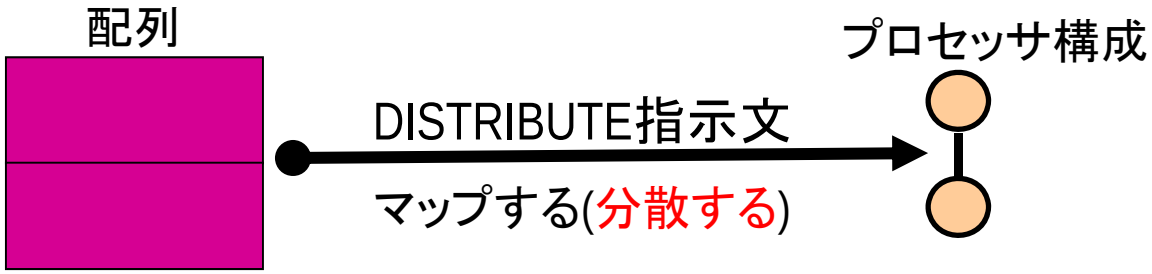
➢多くの場合、PROCESSORS指示文の指定を省略しても特に問題ない

2. HPFの基本機能

プロセッサの構成を宣言する	PROCESSORS指示文
配列を分散する	DISTRIBUTE指示文
並列化可能であることを明示する	INDEPENDENT指示文 [オプション] ◆作業変数指定: NEW節 ◆集計変数指定: REDUCTION節
実行プロセッサを指定する	ON指示構文 [オプション] ◆通信不要を明示: LOCAL節

DISTRIBUTE**指示文**(1)

配列のプロセッサ構成へのマップ方法(**分散**)を指定する



書式: 配列*ti*をプロセッサ構成 *p* 上に分散する

◆1つの配列の分散([ ]内は省略可能)

DISTRIBUTE *t1*(分散方法,...) [ONTO *p*]

◆複数の配列の分散([ ]内は省略可能)

DISTRIBUTE (分散方法,...) [ONTO *p*] :: *t1, t2,...*

分散方法

- 5種類の分散方法が指定可能

  BLOCK、*、GEN_BLOCK、CYCLIC、INDIRECT

DISTRIBUTE**指示文**(2) BLOCK**分散**

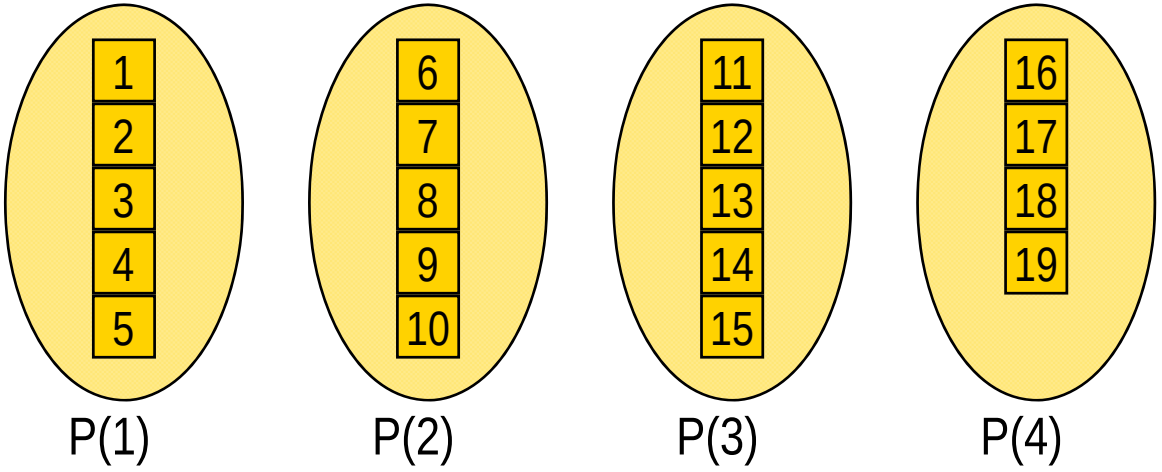
連続した配列要素を均等に分散する

- **分散ブロック幅**= (**配列要素数** – 1) / **プロセッサ数** + 1
- BLOCK(m)**の指定によりm要素ずつの分散も可能**

用途 負荷が要素に対して均一で、近接参照が多い場合に適する

例:

```
!HPF$ PROCESSORS P(4)
      REAL A(19)
!HPF$ DISTRIBUTE A(BLOCK) ONTO P
```



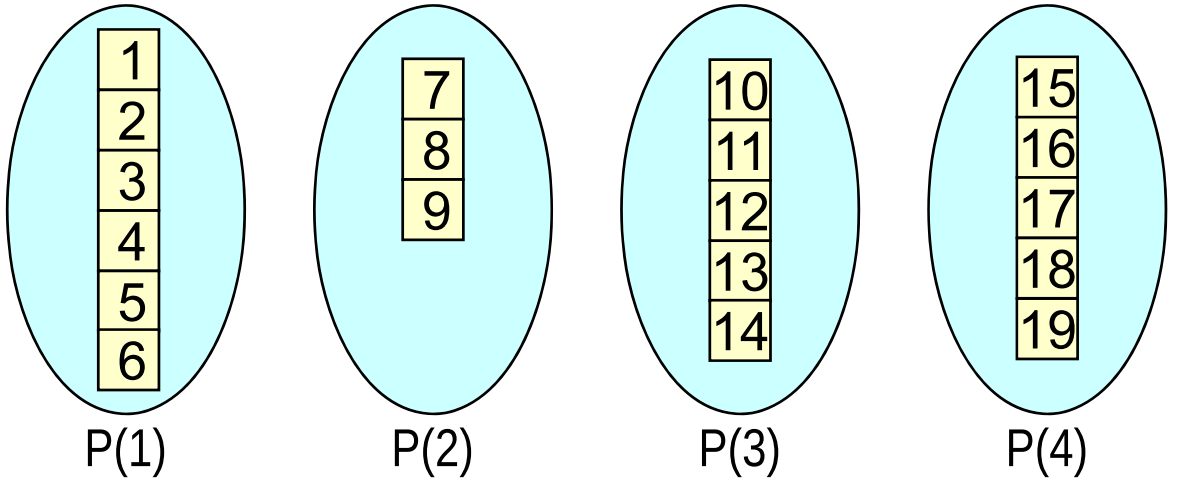
DISTRIBUTE指示文(3) GEN_BLOCK分散

連続した配列要素を不均等に分散する

用途 要素に対する負荷バランスが不均等な場合に適する

例:

```
!HPF$ PROCESSORS P(4)
      REAL A(19)
      INTEGER, PARAMETER:: M(4)=(/6,3,5,5/)
!HPF$ DISTRIBUTE A(GEN_BLOCK(M)) ONTO P
```



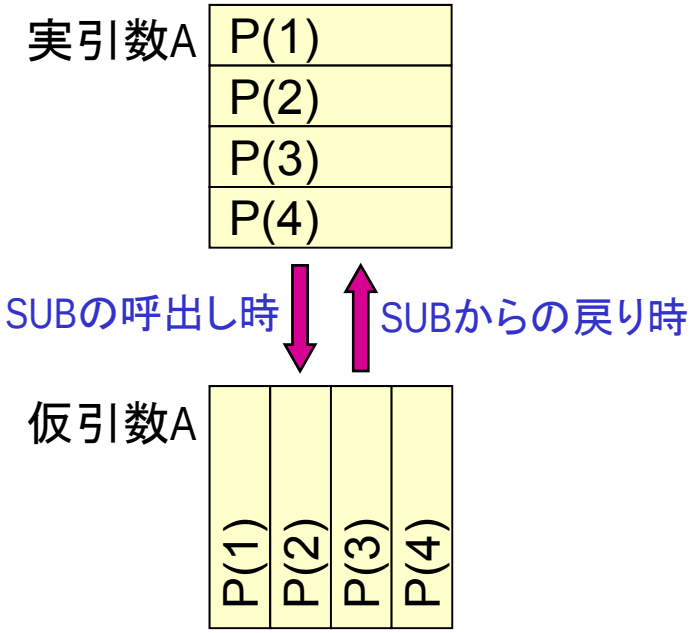
手続境界での引数のマッピング

実引数と仮引数のマッピングは独立に指定可能

例:

```
!HPF$ PROCESSORS P(4)
!HPF$ DISTRIBUTE A(BLOCK,*) ONTO P
      INTEGER A(N,N)
      CALL SUB(A,N)
        .
      END

      SUBROUTINE SUB(A,N)
!HPF$ PROCESSORS P(4)
!HPF$ DISTRIBUTE A(*,BLOCK) ONTO P
      INTEGER A(N,N)
```



実引数と仮引数のマッピングが異なる場合

◆手続の呼出し時・戻り時にそれぞれ自動的に通信が発生する(合計2回)

- ●呼出し時は、実引数のマッピングを仮引数と合わせるため
- ●戻り時は、仮引数のマッピングを実引数のマッピングに戻すため

2. HPFの基本機能

プロセッサの構成を宣言する	PROCESSORS指示文
配列を分散する	DISTRIBUTE指示文
並列化可能であることを明示する	INDEPENDENT指示文 ［オプション］ ◆作業変数指定: NEW節 ◆集計変数指定: REDUCTION節
実行プロセッサを指定する	ON指示構文 ［オプション］ ◆通信不要を明示: LOCAL節

INDEPENDENT指示文(1)

直後のループが並列実行できることを、コンパイラに教える

```
!HPF$ INDEPENDENT
    DO I = 1, N
      A(I)=B(I)+C(I)
    END DO
```

書式: ([ ]内は省略可能)

INDEPENDENT [, NEW(*変数名* ,...)] [, REDUCTION(*変数名*,...)]

オプションとして、以下の2つが指定可能

NEW節	作業変数
REDUCTION節	集計変数

用途

並列化できるにもかかわらず、コンパイラが並列化できないと判定したループだけに指定すればよい (多くの場合、INDEPENDENT指示文なしでも、コンパイラが自動的に並列化可能性を判定できる)

INDEPENDENT指示文(2)

並列化できないループの例 (実行順序により結果が変わる場合)

- ある繰返しで定義された要素を他の繰返しで定義・使用すると並列化を阻害する(データ依存)
- このようなループにはINDEPENDENT指示文は指定できない(指定すると結果不正となる)

DO I = 2,N A(I)=A(I)+A(I-1) END DO	A(2) = A(2) + A(1) ↘ A(3) = A(3) + A(2)	依存: 定義後に使用
DO I = 1,N-1 A(I)=A(I)+A(I+1) END DO	A(1) = A(1) + A(2) ↘ A(2) = A(2) + A(3)	逆依存: 使用後に定義

DO I = 1,N
IF (A(I)>0) S=A(I)
END DO
	S = A(1)
	↓
	S = A(2)
		出力依存:
		定義後に定義

DO I = 1,N
IF (A(I)>0) STOP
END DO
	•ループ中のSTOP文
	•ループからの飛び出し
		制御依存:
		途中で実行が終わってしまう可能性がある

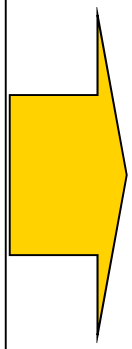
INDEPENDENT指示文(3) REDUCTION節(1)

集計計算	総和・最大・最小のように、どの順序で実行しても結果が(数学的には)同じになる計算
REDUCTION節	集計計算を行うループにINDEPENDENT指示文を指定する場合に、併せて指定する (自動で並列化された場合は不要)

例:

```
!HPF$ DISTRIBUTE A(BLOCK)
      INTEGER A(N),s

並列化対象
      DO I=1,N
        s = s + A(I)
      ENDDO
```



```
!HPF$ DISTRIBUTE A(BLOCK)
      INTEGER A(N),s
!HPF$ INDEPENDENT, REDUCTION(s)
並列化対象
      DO I=1,N
        s = s + A(I)
      ENDDO
```

制限: 集計計算(総和・最大・最小・…)は、以下の形式で記述しなければならない

```
A = A + E        A = E .and. A      A = iand(A,E1,...,En)
A = E + A        A = A .or. E       A = ior(A,E1,...,En)
A = A – E        A = E .or. A       A = ieor(A,E1,...,En)
A = A * E        A = A .eqv. E      A = min(A, E1,...,En)
A = E * A        A = E .eqv. A      A = max(A,E1,...,En)
A = A / E        A = A .neqv. E
A = A .and. E    A = E .neqv. A
```

A: 集計変数
E, Ei: 式

INDEPENDENT指示文(4) REDUCTION節(2)

■集計種別付き REDUCTION節(HPF/JA拡張)

用途 任意の形式の集計計算を含むループに、INDEPENDENT指示文を指定する場合に指定する(自動で並列化された場合は不要)

書式: ([ ]内は省略可能)

REDUCTION(集計種別: *集計変数名* [/ *位置変数名* ,.../],...)

集計種別 +, *, max, min, firstmax, firstmin, lastmax, lastmin, iand, ior, ieor, .and., .or., .eqv., .neqv.

例:

```
!HPF$ DISTRIBUTE A(BLOCK)
      INTEGER A(N)
!配列aの最大値とその場所
      DO i=1,N
        if(a(i).gt.m)then
          m = a(i)
          idx = i
        endif
      ENDDO
```

並列化対象

```
!HPF$ DISTRIBUTE A(BLOCK)
      INTEGER A(N)
!hpf$  independent,reduction(firstmax: m /idx/)
      DO i=1,N
        if(a(i).gt.m)then
          m = a(i)
          idx = i
        endif
      ENDDO
```

並列化対象

2. HPFの基本機能

プロセッサの構成を宣言する	PROCESSORS指示文
配列を分散する	DISTRIBUTE指示文
並列化可能であることを明示する	INDEPENDENT指示文 ［オプション］ ◆作業変数指定: NEW節 ◆集計変数指定: REDUCTION節
実行プロセッサを指定する	ON指示構文 ［オプション］ ◆通信不要を明示: LOCAL節

ON指示構文(1)　実行プロセッサの指定

書式:　*部分配列* がマップされているプロセッサが実行する

◆1つの文・構文に対して([ ]内は省略可能)

ON HOME(*部分配列*) [, LOCAL[(*配列名*)]]

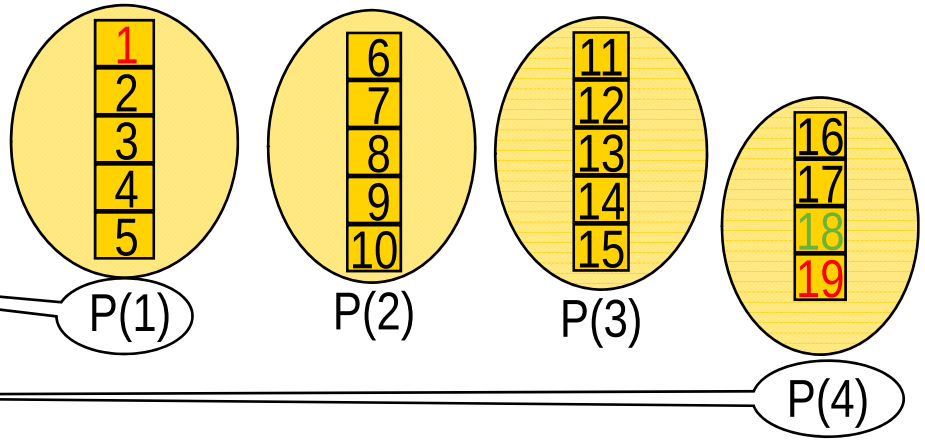
◆複数の文・構文に対して([ ]内は省略可能)

ON HOME(*部分配列*) [, LOCAL[(*配列名*)]] BEGIN
: 実行文・実行構文の列
ENDON

実際上は、LOCALとセットで利用する場合がほとんど

例:

```
!HPF$ PROCESSORS P(4)
      REAL A(19)
!HPF$ DISTRIBUTE A(BLOCK) ONTO P
!HPF$ ON HOME(A(1))
      A(1) = 0
!HPF$ ON HOME(A(19)) BEGIN
      A(18) = 1
      A(19) = 0
!HPF$ ENDON
```

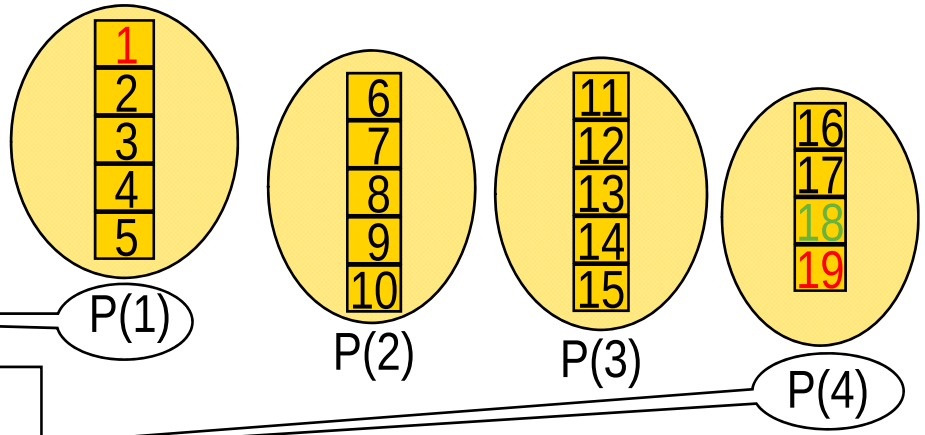


ON指示構文(2)　通信不要の明示

LOCAL節	•指定した変数に対する通信が不要であることをコンパイラに教える •変数指定を省略すると、全ての変数が通信不要であることを意味する
主な用途	•無駄な通信が発生している(コンパイラが通信が不要であることを見切れてない)場合に指定する

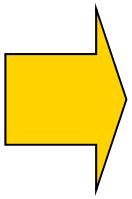
例:

```
!HPF$ PROCESSORS P(4)
      REAL A(19)
!HPF$ DISTRIBUTE A(BLOCK) ONTO P
!HPF$ ON HOME(A(1)), LOCAL(A)
      A(1) = 0
!HPF$ ON HOME(A(19)), LOCAL BEGIN
      A(18) = 1
      A(19) = 0
!HPF$ ENDON
```

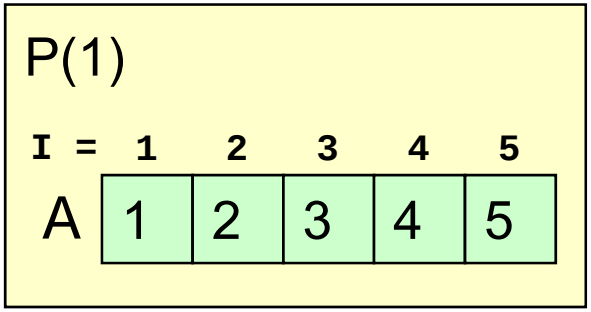


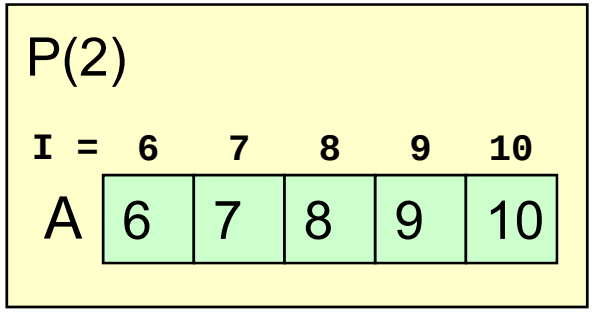
ON指示構文(3) 並列ループへの指定例

```
      REAL A(10)
!HPF$ PROCESSORS P(2)
!HPF$ DISTRIBUTE A(BLOCK) ONTO P
      DO I=1,10
        A(I) = I
      END DO
```



```
      REAL A(10)
!HPF$ PROCESSORS P(2)
!HPF$ DISTRIBUTE A(BLOCK) ONTO P
      DO I=1,10
!HPF$   ON HOME(A(I)), LOCAL BEGIN
        A(I) = I
!HPF$   END ON
      END DO
```





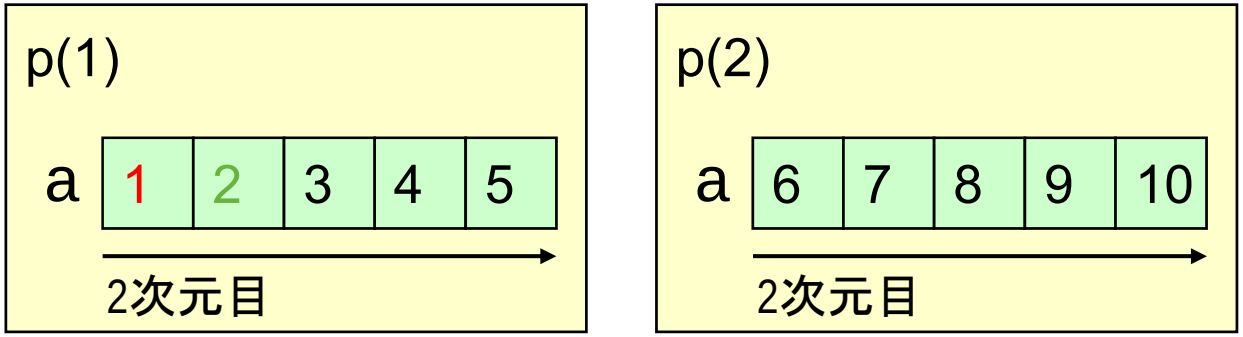
- DO Iの繰返しは、A(I)を保持するプロセッサが実行すると、通信が不要であることを指定する
- 通常は、コンパイラが自動的に通信が最小になるよう並列化するため不要。無駄な通信が発生している場合に指定する。

ON指示構文(4) 境界処理ループへの指定例

```
      real a(10,10)
!hpf$ PROCESSORS p(2)
!hpf$ DISTRIBUTE (*,BLOCK) ONTO p :: a
      do i=1,10
        a(i,1)=a(i,2)
      enddo
```

```
      real a(10,10)
!hpf$ PROCESSORS p(2)
!hpf$ DISTRIBUTE (*,BLOCK) ONTO p :: a
!hpf$ ON HOME(a(:,1)), LOCAL BEGIN
      do i=1,10
        a(i,1)=a(i,2)
      enddo
!hpf$ ENDON
```

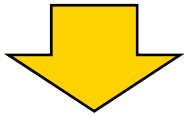
分散されていない次元は":"を指定する



- 例えば、2プロセッサで実行する場合、a(:,1)を保持するプロセッサ(p(1))だけで実行すると、通信なしで実行可能。これをLOCAL節で明示する。(一方、10プロセッサで実行する場合は通信が必要なので、LOCAL節は指定できない)
- ON指示構文とLOCAL節の指定がないと、p(2)も実行に参加し、通信が発生する

3. HPFの基本機能まとめ

プロセッサ構成を宣言する	PROCESSORS指示文
配列を分散する	DISTRIBUTE指示文
並列化可能であることを明示する	INDEPENDENT指示文 [オプション] ◆作業変数指定: NEW節 ◆集計変数指定: REDUCTION節
実行プロセッサを指定する	ON指示構文 [オプション] ◆通信不要を明示: LOCAL節



規則的な問題なら、大半のプログラムがこれだけで並列化可能

3. HPFの基本機能まとめ

HPFプログラミングの基本手順

1. コードのクリーンナップ
   - ベクトル化のチューニングは、並列化前に済ませておく
2. どのループを並列化するか決める
   - 一番実行コストの高いループ
3. 配列のマッピングを決める
   - 並列化するループに対応する次元でマップする
4. HPFコンパイラで翻訳してみる
5. 並列化情報リスト・診断メッセージから指示文を追加・修正する
   - 配列のマッピングを追加・修正する
   - 並列化可能であることを明示する指示文(INDEPENDENT)を追加する
   - 通信が不要であることを明示する指示文(ON+LOCAL)を追加する
   - その他の指示文や、F90/SX指示行を追加する
6. 実行性能を確認し、上記の手順4・5を繰り返す

4. HPFプログラミング演習(1)

姫野ベンチマークコード(himenoBMTsp_s.f)**を用いて，**HPF**プログラミングの流れを学習する.**

※姫野ベンチマーク：ポアッソン方程式解法をヤコビの反復法で解く場合の主要なループの処理速度を計るものである(http://accc.riken.jp/2145.htm).

※本講習では上記 Web**ページからダウンロードしたソースコードの一部修正したものを用いる.**

4. HPFプログラミング演習(1) Step-1

プログラムをコンパイルして実行する

- 姫野ベンチマークコードをhimeno.fとして用意.
- himeno.fはFortranで記述した逐次実行プログラムなのでFORTRAN90/SXコンパイラでコンパイルする.
- ディレクトリを移動する.

  % cd HPF/practice_1

- FTRACE情報を採取して演算コストを把握する.

  % sxf90 -ftrace –R2 himeno.f

- 実行スクリプトを用意していますので，サブミットしてSX-ACEで実行する.

  % qsub run.sh

4. HPFプログラミング演習(1) Step-1

実行結果を確認

- 実行結果はpractice_1.oXXXX(XXXXはジョブID)として格納される.

% cat practice_1.oXXXX

```
mimax=  513   mjmax=  257   mkmax=  257
imax=  512   jmax=  256   kmax=  256
Start rehearsal measurement process.
Measure the performance in 1000 times.
 MFLOPS:  26981.75    time(s):  41.46163582801819   0.2500000
```

- FTRACE情報を確認する.

% sxftrace –f ftrace.out

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.%
jacobi	1	41.456(99.8)	41456.289	60596.7	27385.2	99.57	253.4	41.456	0.000	0.000	3.191	8.065	46.74
initmt	1	0.092(0.2)	91.688	11472.6	0.7	99.32	256.0	0.092	0.000	0.000	0.037	0.000	0.00
unnamed$main	1	0.004(0.0)	3.968	8915.8	0.6	95.09	254.2	0.002	0.000	0.000	0.000	0.000	0.00
pghpf$static$init	1	0.000(0.0)	0.003	133.3	0.0	0.00	0.0	0.000	0.000	0.000	0.000	0.000	0.00
total	4	41.552(100.0)	10387.987	60483.4	27322.1	99.57	253.4	41.550	0.000	0.001	3.228	8.065	46.74

4. HPFプログラミング演習(1) Step-2

演算コストが集中しているのはサブルーチンjacobi

- サブルーチンjacobiのどのループを並列化するか検討する.

```
  176: +------>          DO loop=1,nn
  177: |+----->             DO K=2,kmax-1
  178: ||+---->                DO J=2,jmax-1
  179: |||V--->                   DO I=2,imax-1
  180: ||||    A                     S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
  181: ||||                1                +a(I,J,K,3)*p(I,J,K+1)
  182: ||||                2                +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
  183: ||||                3                -p(I-1,J+1,K)+p(I-1,J-1,K))
  184: ||||                4                +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
  185: ||||                5                -p(I,J+1,K-1)+p(I,J-1,K-1))
  186: ||||                6                +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
  187: ||||                7                -p(I+1,J,K-1)+p(I-1,J,K-1))
  188: ||||                8                +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
  189: ||||                9                +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
  190: ||||    A                     SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
  191: ||||                          GOSA=GOSA+SS*SS
  192: ||||    A                     wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
  193: |||V--                     enddo
  194: ||+---                  enddo
  195: |+----                enddo
  197: |+----->             DO K=2,kmax-1
  198: ||+---->                DO J=2,jmax-1
  199: |||V--->                   DO I=2,imax-1
  200: ||||    A                     p(I,J,K)=wrk2(I,J,K)
  201: |||V--                     enddo
  202: ||+---                  enddo
  203: |+----                enddo
  205: +------           enddo
```

4. HPFプログラミング演習(1) Step-2

データマッピングの検討

- 176-205行目の最外ループを並列化すると配列Pが正しく計算されない(配列Pはループ変数loopの次元を持たないので，並列化すると配列Pに逐次計算と同じ結果が格納されない).
- 177-195行目と197-203行目のループを並列化の対象とする.
- ループ変数kの次元を有する配列がマッピングの対象となる.

配列の宣言内容	配列名
(mimax,mjmax,mkmax)	p,bnd,wrk1,wrk2
(mimax,mjmax,mkmax,4)	a
(mimax,mjmax,mkmax,3)	b,c

- DISTRIBUTE指示文を挿入する(メインプログラムとサブルーチンinitに対して同様に挿入する). エディタを用いてhimeno.fを修正する.

  % vi himeno.f (エディタはviの他にも利用可能)

4. HPFプログラミング演習(1) Step-2

- DISTRIBUTE指示文の挿入.

```
 common /pres/ p(mimax,mjmax,mkmax)
       common /mtrx/ a(mimax,mjmax,mkmax,4),
      +       b(mimax,mjmax,mkmax,3),c(mimax,mjmax,mkmax,3)
       common /bound/ bnd(mimax,mjmax,mkmax)
       common /work/ wrk1(mimax,mjmax,mkmax),wrk2(mimax,mjmax,mkmax)
CC other constants
       common /others/ imax,jmax,kmax,omega
!HPF$ DISTRIBUTE (*,*,BLOCK):: p,bnd,wrk1,wrk2
!HPF$ DISTRIBUTE (*,*,BLOCK,*) :: a,b,c
```

- 177-195行目のループには変数gosaの総和を計算する処理があるので,INDEPENDENT指示文でREDUCTION指定を行う.

```
!HPF$ INDEPENDENT,REDUCTION(gosa)
          DO K=2,kmax-1
             DO J=2,jmax-1
                DO I=2,imax-1
                   S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
            (中略)
                   GOSA=GOSA+SS*SS
                   wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
                enddo
             enddo
          enddo
```

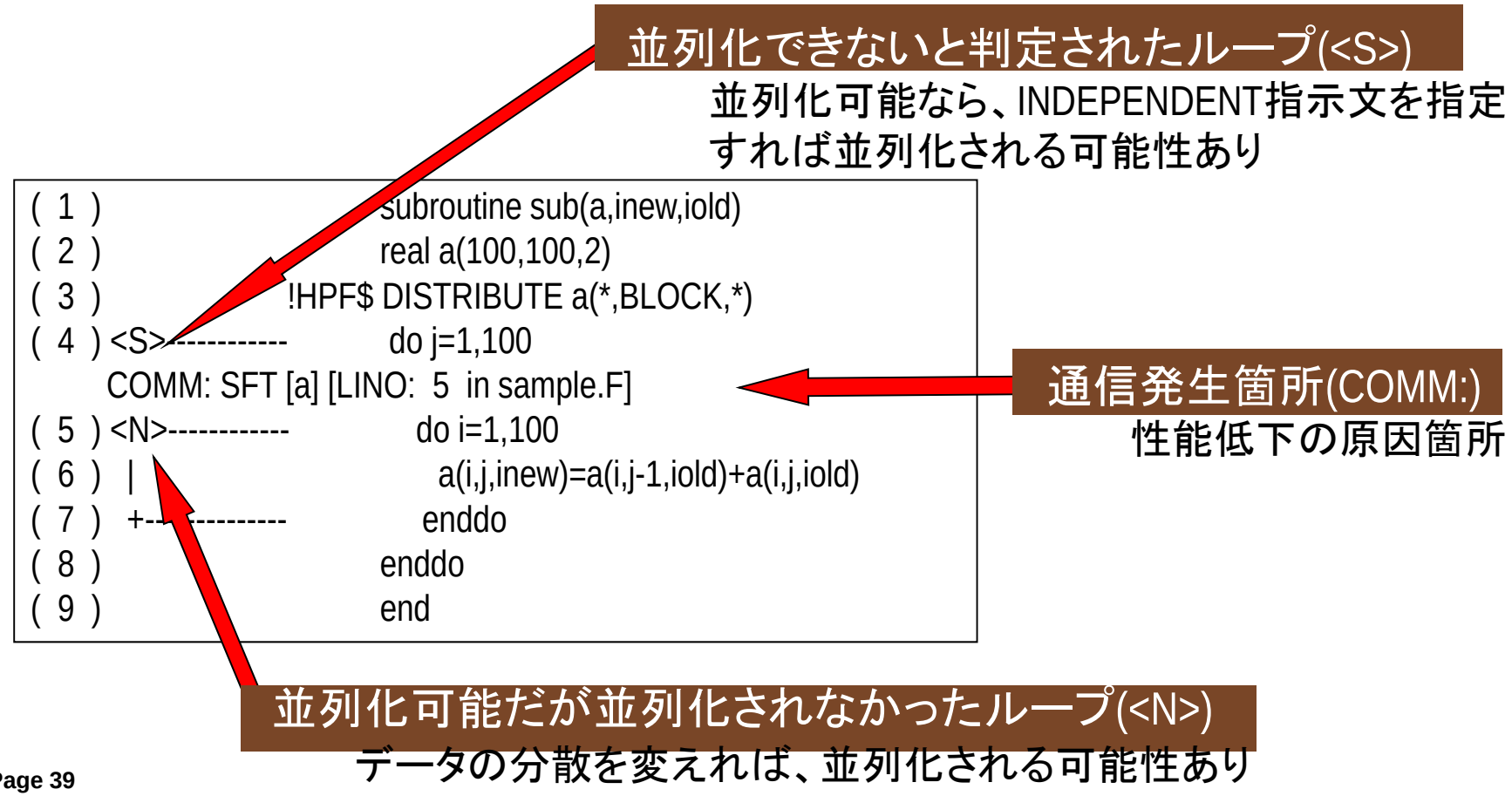
総和計算

4. HPFプログラミング演習(1) Step-3

- **合計7行の指示文を追加したら，コンパイルを実行する．**-Mlist2**オプションを追加して並列化やデータ転送の有無を確認する．**

  % sxhpf -Mlist2 himeno.f

- (**参考**)-Mlist2**オプションによる並列化情報リスト**filename.lst**の見方**



4. HPFプログラミング演習(1) Step-3

himeno.lstを確認

```
(  182) <S>------------        DO loop=1,nn
(  183)                    !HPF$ INDEPENDENT,REDUCTION(gosa)
        COMM: RED [gosa] [LINO: 184 in himeno-2.f]
        COMM: SFT [p] [LINO: 184 in himeno-2.f]
        HOME: a(:,:,k,:)
(  184) <P>------------           DO K=2,kmax-1
(  185)  |<I>-----------              DO J=2,jmax-1
(  186)  |<I>-----------                 DO I=2,imax-1
(  187)  |                                 S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
(  188)  |                       1               +a(I,J,K,3)*p(I,J,K+1)
(  189)  |                       2               +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
(  190)  |                       3               -p(I-1,J+1,K)+p(I-1,J-1,K))
(  191)  |                       4               +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
(  192)  |                       5               -p(I,J+1,K-1)+p(I,J-1,K-1))
(  193)  |                       6               +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
(  194)  |                       7               -p(I+1,J,K-1)+p(I-1,J,K-1))
(  195)  |                       8               +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
(  196)  |                       9               +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
(  197)  |                                 SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
(  198)  |                                 GOSA=GOSA+SS*SS
(  199)  |                                 wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
(  200)  |                              enddo
(  201)  |                           enddo
(  202)  +------------            enddo
        HOME: p(:,:,k)
(  204) <P>------------           DO K=2,kmax-1
(  205)  |<I>-----------              DO J=2,jmax-1
(  206)  |<I>-----------                 DO I=2,imax-1
(  207)  |                                 p(I,J,K)=wrk2(I,J,K)
(  208)  |                              enddo
(  209)  |                           enddo
(  210)  +------------            enddo
(  212)                          enddo
```

4. HPFプログラミング演習(1) Step-4

HPFプログラムの実行

- 実行用スクリプトをrun2.shとして用意している(4プロセスでの実行).

  % qsub run2.sh

- 実行結果を確認する. 実行結果はstep-4.oXXXX(XXXXにはジョブIDが入ります)として格納される.

  % cat step-4.oXXXX

```
mimax=  513   mjmax=  257   mkmax=  257
imax=  512   jmax=  256   kmax=  256
Start rehearsal measurement process.
Measure the performance in 1000 times.
 MFLOPS:  87137.26    time(s):  12.83845210075378  0.6827177
```

➢ 41.5秒が12.8秒に短縮.

5. 並列化情報リストの利用

■ 並列化情報リストから問題を抽出する(1)

◆ 並列化できないと判定されたループの抽出方法

```
%>grep “<S>” filename.lst
(  57) <S>----------    do ij=2,ibar*2,2
```

57行目のdo ijのループは並列化できないと判定された

- 特に、コストの高い通信を伴う場合、注意が必要。INDEPENDENT指示文(+NEW節、REDUCTION節)を指定すると並列化される場合が多い
- 同じループネストに <P> (並列化された)マークのついたループがあり、通信がない場合は問題ない

```
(131) <P>----------          do k=1,ibar
(132)  |  <S> ----------        do n=1,n
```

◆ 並列化可能だが並列化されなかったループの抽出方法

- ループ中の配列を分散すると、並列化される場合あり
- 通信を伴わない場合は、問題ないことが多い

```
%>grep “<N>” filename.lst
(  44) <N>----------    do i=0,ib
```

44行目のdo iのループは並列化できるが並列化しなかった

5. 並列化情報リストの利用

■ 並列化情報リストから問題を抽出する(2)

◆ 通信発生箇所の抽出方法

```
%> grep  COMM:  finename.lst
COMM: SCL [u] [LINO: 137 in filename.hpf]
```

◆ 出力フォーマット

COMM: **通信パタン** [**変数名**] [LINO: **行番号** in **ファイル名**]

◆ 通信パタンとしては、以下のものがある

・ マッピング(DISTRIBUTE**指示文**等)の追加・修正やINDEPENDENT**指示文**による並列化、ON**指示構文**+LOCAL**節**による通信抑制などで削減できる場合も多い

SFT	シフト通信。コストが低いので通常は問題ない。
RED	集計計算。並列ループの外側に生成されているなら通常やむを得ない。 並列ループの内側ならREDUCTION**節付き**INDEPENDENT**指示文**が有効。
SCL	一要素通信。配列が対象の場合は、通信コストが高いため**チェックが必要。**
CPY	**配列のテンポラリへのコピー。**通信対象の配列と、ループ並列化の 基準配列との間にマッピングの不整合がある場合等に発生する。
G/S	Gather/Scatter。間接アクセスループ等で発生し**通信コストが高い**場合が多い

6. クリーンナップとデバッグ

補足情報として，HPFプログラムのクリーンナップとデバッグについて紹介する．

マップできない配列

- 他の変数とメモリ領域を共有する配列はマップできない。(翻訳時にエラーとなる)

◆ EQUIVALENCE文に記述した配列

修正方法:

作業配列の利用・再利用や、動的に大きさを決めたい配列には、割付け配列や自動割付け配列を使う

ある型の配列を別の型で参照するようなプログラミングは避ける。

◆ POINTER/TARGET属性を持つ配列

修正方法:

動的にメモリを確保したい場合には、
割付け配列や自動割付け配列を利用する。

●自動割付け配列とは:

```
SUBROUTINE SUB(N)
 COMMON /COM/ M
 REAL A(N,M)  ！ 大きさが、引数や共通ブロック変数により実行時に決まる局所配列
```

Fortranとの非互換事項(1)

- 手続間で結合する配列の形状・型等は、原則として同一にする
- HPFの文法違反の例

◆ 引数結合(実引数と仮引数)

例1: アドレス渡し

```
        real a(100,100)
!hpf$ distribute a(*,block)
        call sub(a(1,i))
        end
        subroutine sub(a)
        real a(100)
```

例2: 実引数と仮引数の形状が異なる

```
        real a(10000)
        call sub(a, 10)
        end
        subroutine sub(a,n)
        real a(n,n)
!hpf$ distribute a(*, block)
```

◆ 共通ブロック

例1: マッピング指定漏れ

```
        common /com/ a(100,100)
!hpf$ distribute a(*,block)
        end
        subroutine sub
        common /com/a(100,100)
```

例2: 使わない手続で宣言省略

```
        common /com/ a(100),b(100)
!hpf$ distribute (block) :: a,b
        end
        subroutine sub
        common /com/a(100)
!hpf$ distribute (block) :: a
```

引数結合(1) HPF/SX V2のデバッグ機能

■実行時に、エラーが発生した手続や対象引数の情報が標準エラー出力に出力される

例1: アドレス渡し

```
      real a(100,100)
!hpf$ distribute a(*,block)
      call sub(a(1,i))
      end
      subroutine sub(a)
      real a(100)
```

例2: 実引数と仮引数の形状が異なる

```
      real a(10000)
      call sub(a,10)
      end
      subroutine sub(a,n)
      real a(n,n)
!hpf$ distribute a(*,block)
```

例1: ″仮引数名″: Nonsequential dummy array is associated with array element or scalar actual. PROG=″手続名″ ELN=″行番号″

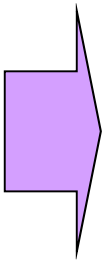
例2: ″仮引数名″: Dummy argument rank differs from actual. PROG=″手続名″ ELN=″行番号″

引数結合(2) 修正方法1

対応する実引数と仮引数の形状や型等は一致させる

例1:

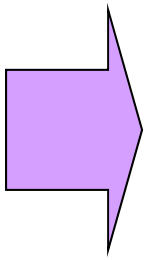
```
        real a(100,100)
!hpf$ distribute a(*,block)
        do I=1,100
           call sub(a(1,I)) ! アドレス渡し
        enddo
        end
        subroutine sub(a)
        real a(100)
```



```
        real a(100,100)
!hpf$ distribute a(*,block)
        do I=1,100
           call sub(a(:,I)) ! 部分配列
        enddo
        end
        subroutine sub(a)
        real a(100)
```

例2:

```
real a(100000)    ! 大きめに
call sub(a,10)
end
subroutine sub(a,n)
real a(n,n)
```



```
real,allocatable :: a(:,:) ! 割付け配列
read(5) n
allocate(a(n,n))  !使用する形状で割付け
call sub(a,n)
end
subroutine sub(a,n)
real a(n,n)
```

引数結合(3)　修正方法2

■大きさ引継ぎ配列(擬寸法配列)は、最後の次元の寸法がないためマップしたり、マップされた実引数と結合できない

➡ 整合配列、又は形状引継ぎ配列を利用する

例:

```
real a(100,100)
call sub(a)
  :
end

subroutine sub(a)
real a(100,*)
```

整合配列

```
real a(100,100)
call sub(a,100)
  :
end

subroutine sub(a,n)
real a(100,n)
```

形状引継ぎ配列:
Interfaceブロック等により、呼出し側で、呼ばれ側手続の引数が形状引継ぎ配列であることを明示する必要がある(Fortranの仕様)

```
real a(100,100)
interface
  subroutine sub(a)
  real a(:,:)
  end subroutine
end interface
call sub(a)
  :
end

subroutine sub(a)
real a(:,:)
```

共通ブロック(1) HPF/SX V2のデバッグ機能

翻訳時オプション-Mcommonchkを指定して翻訳すると、
実行時に不整合が検出され、
修正すべき手続名や配列名が標準エラー出力に表示される

注意 本オプションを指定する場合、全ての手続に対して指定する

例1: マッピング指定漏れ

```
      common /com/ a(100,100)
!hpf$ distribute a(*,block)
      end
      subroutine sub
      common /com/a(100,100)
```

例2:使わない手続で宣言省略

```
      common /com/a(100),b(100)
!hpf$ distribute (block) :: a,b
      end
      subroutine sub
      common/com/ a(100)
!hpf$ distribute (block) ::a
```

例1:

Inconsistency detected in the mapping attribute of common variable between “手続名” and “手続名” : “配列名” in /共通ブロック名/

例2:

Inconsistency detected in the number of explicitly mapped arrays of common block between “手続名” and “手続名” : “配列名” in /共通ブロック名/

共通ブロック(2)　修正方法

- 共通ブロックの宣言は、全手続で一致させる(マッピングを含む)
- 大域変数はモジュールやINCLUDEファイル中に宣言すると、
  宣言の記述が一回で済むため、書き忘れや書き誤りが防げる

例：･モジュールによる大域変数の宣言

```
        module com1
        dimension a(100,100)
!hpf$ distribute (*,block) :: a
        end module
        module com2
        dimension b(100,100),c(100,100)
!hpf$ distribute (*,block) :: b,c
        end module


        subroutine sub()
        use com1
        use com2
        end
        subroutine sub2()
        use com1
        use com2
        end
```

•INCLUDEファイルによるCOMMONの宣言

```
%> cat  com1.h
        common /com1/ a(100,100)
!hpf$ distribute (*,block) :: a
%> cat  com2.h
        common /com2/ b(100,100),c(100,100)
!hpf$ distribute (*,block) :: b,c
```

```
        subroutine sub()
        include “com1.h”
        include “com2.h”
        end
        subroutine sub2()
        include “com1.h”
        include “com2.h”
        end
```

Fortranとの非互換事項(2)

■ 宣言範囲外アクセス

- ◆ プログラムミスの他、ベクトル長を稼ぐためにループを一重化する場合などにも発生する

例: ループ一重化

```
      real a(100,100)
!hpf$ distribute a(*,block)
      do i=1,10000
        a(i,1) = ...
      enddo
```

・分散次元より前の場合は可能

```
      real a(100,100,100)
!hpf$ distribute a(*,*,block)
      do k=1,100
      do i=1,10000
        a(i,1,k) = ...
      enddo
      enddo
```

- ◆ システム領域が破壊され、全く別の場所や、ずっと後のステップでアボートする場合がある。

宣言範囲外アクセス：HPF/SX V2のデバッグ機能

■ 宣言範囲外アクセス検出機能

翻訳時オプション-Msubchkを指定して翻訳すると、
実行時に宣言範囲外アクセスの有無がチェックされる

並列化・ベクトル化は可能な限り行われる

範囲外アクセス検出時、配列名、行番号等の情報が警告又は致命的
エラーメッセージとして標準エラー出力に出力される(既定値は警告)。

- 実行時オプション-hpf –subchk fatal を指定すると、
  範囲外アクセス検出時に実行を終了させることもできる。

例：•実行時オプションの指定方法： mpirun –nn 2 –nnp 4 ./a.out –hpf –subchk fatal

```
        real a(100,100)
!hpf$ distribute a(*,block)
        do i=1,10000
          a(i,1) = ...
        enddo
```

“配列名” is accessed out of declared bounds along %d th dim
PROG=“手続名” ELN=“行番号”

7. 実行時性能情報の取得方法

■ 主に3つの手段がある

MPIPROGINF: プログラム全体の情報

◆特別な翻訳時オプションは不要。**測定オーバヘッドなし。**

◆使い方はMPIの場合と同じ。

●実行時環境変数MPIPROGINFを指定して実行する

ftrace: 手続毎の情報

◆翻訳時オプション-ftraceを指定して実行する

◆使い方はMPIの場合と同じ

```
%>sxftrace –f ftrace.out.*
```

etime: 計時用手続。任意の区間の実行時間（経過時間）

◆使い方はFortranの場合と同じ

◆並列ループ中から呼び出すことは出来ない

8. HPFプログラミング演習(2)

3つのプログラム単位：全86行(空白行とコメントを含む)

- モジュールparam
  - 問題サイズと時間発展ループの繰返し数を大域定数として宣言
- 主プログラムsample
  - 主要配列は2次元: a(n,n),b(n,n),c(n,n)
  - 初期化後、時間発展ループ中で計算を行い、実行時間とチェックsumを出力
- サブルーチンbound
  - 境界処理ループだけを含む

備考: 頻出パタンの練習です。計算内容に意味はありません。

既にコードのクリーンナップ後の状態

- このままでもHPFコンパイラで翻訳、実行は可能(もちろん、何台で実行しても速くはならない)

8. HPFプログラミング演習(2)

プログラムをコンパイルして実行する

- ディレクトリを移動する.

  % cd HPF/practice_2

- FTRACE情報を採取して演算コストを把握する.

  % sxf90 -ftrace sample.F

- 実行スクリプトを用意していますので，サブミットしてSX-ACEで実行する.

  % qsub run.sh

- 実行結果を確認する．実行結果はpractice_2.oXXXX(XXXXはジョブID)として格納される.

  % cat practice_2.oXXXX

8. HPFプログラミング演習(2)

- もっともコストの高いループは以下.

```
! main loop
      do j=2, n-1
         do i=2, n-1
            ix = idxx(i)
            iy = idxy(j)
            a(i,j) = (b(i,j)+b(i-1,j)+b(i+1,j)+b(i,j-1)+b(i,j+1))*0.2d0*c(ix,iy)*ap
         enddo
      enddo
```

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.%
sample	1	5.370(100.0)	5370.141	20635.7	7037.1	99.74	253.3	5.370	0.000	0.000	0.013	2.588	28.53
bound	1002	0.001(0.0)	0.001	12982.4	0.0	98.58	255.9	0.000	0.000	0.000	0.000	0.000	50.83
total	1003	5.371(100.0)	5.355	20634.7	7036.3	99.74	253.3	5.370	0.000	0.000	0.013	2.588	28.53

8. HPFプログラミング演習(2)

並列化するループ(35行目)中で定義される配列をマップする

- do jのループがアクセスする、配列aの2次元目をマップする
  - 通常は、BLOCK分散が効率が良い
  - 問題サイズの関係上、BLOCK分散では負荷バランスが不均等になる場合等、GEN_BLOCK分散が有効な場合もある。
  - 本ループだけを見れば、右辺の配列b,cはマップする必要はない。但し、マップしても通信が出ない場合は、マップしたほうがメモリの節約になる

```
!hpf$ distribute (*,block) :: a
                :
! main loop
        do j=2, n-1
          do i=2, n-1
            ix = idxx(i)
            iy = idxy(j)
            a(i,j) = (b(i,j)+b(i-1,j)+b(i+1,j)+b(i,j-1)+b(i,j+1))*0.2d0*c(ix,iy)*ap
          enddo
        enddo
```

8. HPFプログラミング演習(2) アクセスパタン表

- 各配列に対して、並列化対象ループがアクセスする次元の、添字の一覧表を作成すると、最適なデータマッピングの候補を決めるのに便利
- **例:**

```
do j=2, n-1  ！ 並列化対象
  do i=2, n-1
    ix = idxx(i)
    iy = idxy(j)
    a(i,j) = (b(i,j)+b(i-1,j)+b(i+1,j)+b(i,j-1)+b(i,j+1))*0.2d0*c(ix,iy)*ap
  enddo
enddo
```

参照種別＼配列名	a	b	c	idxx	idxy
定義	j				
使用		j, j-1, j+1	*	*	j

- 並列化対象ループ(do j)がアクセスする次元の添字を、定義と使用に分けて一覧に。
- 配列cやidxxは、並列化対象DOループのDO変数jに対応する次元がないため、「＊」と表示。

既にデータマッピングを決定した配列aを基準とすると

- **配列bは、**
  - ALIGN b(j) WITH a(j)
  - ALIGN b(j) WITH a(j-1)
  - ALIGN b(j) WITH a(j+1)
  - **非分散**

  のいずれかが最適
- **配列idxyは、**
  - ALIGN idxy(j) WITH a(j)
  - **非分散**

  のいずれかが最適
- **配列c、idxxは、**
  - **非分散が最適**

※実際のALIGNの指定にはテンプレートが必要

8. HPFプログラミング演習(2)

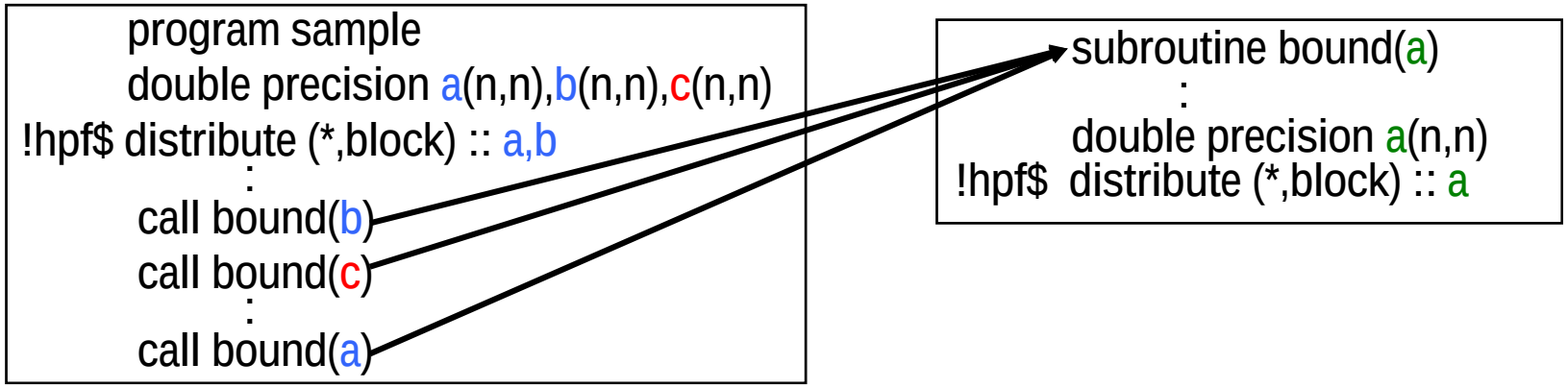
- マップした配列aが出現するループを探す(50行目)
  - 左辺に出現する配列bを配列aに合わせてマップする
    - 分散次元の宣言範囲が同一で、アクセスにもずれがない(共にj)ため、distribute指示文だけでよい
    - もちろん、align指示文やtemplate指示文を利用して、同等のマッピングを指定することも可能

```
        double precision a(n,n),b(n,n)
!hpf$ distribute (*,block) :: a,b
              :
       do j=2, n-1
         do i=2, n-1
           ix = idxx(i)
           b(ix,j) = a(i,j) * c(i,j)
           ap = ap * a(i,j)
         enddo
       enddo
```

- ここで、配列bのデータマッピングは、前頁のアクセスパタン表から導出される候補にも適合していることに注意。

8. HPFプログラミング演習(2)

- 仮引数をマップした手続boundの呼出箇所をチェックする
  - 実引数が配列aの場合(48行目)は、仮引数とマッピングが同一



  - 実引数が配列cの場合(28行目)は、配列cがマップされていないので、サブルーチンboundの呼出し時と戻り時に通信が発生
    - 実配列引数cを仮引数に合わせてマップするか、サブルーチンboundのクローニングを行うかを決める。
    - この作業を忘れた場合、実行時オプション-hpf –commmsgを付けて実行すると、手続呼出し時に通信/コピーが発生する際、メッセージが出力されるのでチェック可能。

8. HPFプログラミング演習(2)

- 配列cはメインループ中で間接アクセスされているので、サブルーチンboundの仮引数に合わせてマップすると、配列aやbとアクセスパタンが一致せず通信が発生
  - 配列cはマップしない（非分散とする。p.5のアクセスパタン表から導出されるマッピングの候補にも適合していることに注意）
  - サブルーチンboundのクローニングを行う

```
      double precision a(n,n),b(n,n),c(n,n)
!hpf$ distribute (*,block) :: a,b
        :
! main loop
      do j=2, n-1
        do i=2, n-1
          ix = idxx(i)
          iy = idxy(j)
          a(i,j) = (b(i,j)+b(i-1,j)+b(i+1,j)+b(i,j-1)+b(i,j+1))*0.2d0*c(ix,iy)*ap
        enddo
      enddo
```

8. HPFプログラミング演習(2)

- 仮引数をマップしていないこと以外、サブルーチンboundと同一のサブルーチンbound_nodistを作成し、boundの実引数が配列cの場合(28行目)、bound_nodistの呼出しに置き換える

```
        double precision a(n,n),b(n,n),c(n,n)
!hpf$ distribute (*,block) :: a,b
        :
        call bound(b)
        call bound_nodist(c)
        :
        call bound(a)
        :
        end
        subroutine bound(a)
        double precision a(n,n)
!hpf$   distribute (*,block) :: a
        :
        end
        suboutine bound_nodist(a)
        double precision a(n,n)
        :
        end
```

8. HPFプログラミング演習(2)

- -Mftn、–Mlist2、–Minfoオプション付きで翻訳する
  - 並列化情報リストをチェックする (-Mlist2オプションで生成される)
    - -Minfo:診断メッセージを出力、-Mftn:F90/SXによる翻訳・リンクを省略

```
%> sxhpf  –Mftn  –Mlist2  –Minfo  sample.F
                    :
%> grep  “<S>”  sample.lst
(    35) <S>----------------     do iter=1,maxiter
(    50) <S>----------------        do j=1,n
(    51) <S>----------------           do i=1,n
%> grep  COMM:  sample.lst
          COMM:  SFT [b]  [LINO:  38  in sample.F]
          COMM:  CPY [a]  [LINO:  54  in sample.F]
          COMM:  SCL [a]  [LINO:  54  in sample.F]
          COMM:  RED [bp]  [LINO:  55  in sample.F]
          COMM:  RED [sum]  [LINO:  64  in sample.F]
          COMM:  SFT [a]  [LINO:  83  in sample.F]
          COMM:  SFT [a]  [LINO:  83  in sample.F]
```

並列化できないと判定されたループの抽出

時間発展ループ。並列化できない。

50,51行目のループネストは要チェック

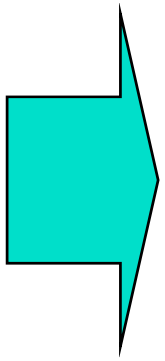
通信発生箇所の抽出
- SFT(シフト)は多くの場合問題なし
- RED(集計)は多くの場合やむを得ない

要チェック。上記の50、51行目の<S>に対応して出ている可能性あり
- CPY(コピー)は、テンポラリへの置き換え(通常通信を伴う)
- SCL(要素毎)は、配列が対象の場合、高コスト

8. HPFプログラミング演習(2)

- 配列a,bの分散次元に対応するdo jで並列化可能だが、コンパイラは、自動判定できていない
  - do jのループにindependent指示文を指定する
  - 但し、変数apに対する集計計算(総積)を含むため、reduction節も必要
    - reduction節を忘れると結果不正となる

```
・並列化情報リスト
                        !hpf$  distribute (*,block) :: a,b
                                          :
(    50) <S>------------          do j=1,n
         COMM:  CPY [a] [LINO:  54  in sample.F]
         COMM:  SCL [a] [LINO:  54  in sample.F]
(    51) <S>------------              do i=1,n
         COMM:  RED [ap] [LINO:  55  in sample.F]
(    52)                                ix = idxx(i)
(    53)                                b(ix,j) = a(i,j)*c(i,j)
(    54)                                ap = ap * a(i,j)
(    55)                              enddo
(    56)                            enddo
```



```
!hpf$  distribute (*,block) :: a,b
                :
!hpf$ independent, reduction(ap)
          do j=1,n
             do i=1,n
                ix = idxx(i)
                b(ix,j) = a(i,j)*c(i,j)
                ap = ap * a(i,j)
              enddo
          enddo
```

8. HPFプログラミング演習(2)

- 再度、-Mftn、–Mlist2、–Minfoオプション付きで翻訳する
  - ◆ 並列化情報リストをチェックする

```
%> sxhpf  –Mftn  –Mlist2  –Minfo  sample.F
                    :
%> grep  “<S>”  sample.lst
(    35) <S>----------      do iter=1,maxiter
(    55) <S>----------           do i=1,n
%> grep “<P>”  sample.lst
(    38) <P>----------           do j=2,n-1
(    47) <P>----------           do i=1,n
(    54) <P>----------           do j=1,n
(    65) <P>----------         do  j=1,n
%> grep  COMM:  sample.lst
          COMM:  SFT [b]  [LINO:  38  in sample.F]
          COMM:  RED [bp]  [LINO:  54  in sample.F]
          COMM:  RED [sum]  [LINO:  65  in sample.F]
          COMM:  SFT [a]  [LINO:  84  in sample.F]
          COMM:  SFT [a]  [LINO:  84  in sample.F]
```

並列化できないと判定されたループの抽出

時間発展ループ。並列化できない。

同じループネストの54行目の<S>が消え、<P>に変わっており、重い通信も出ていないので問題なし

並列化されたループの抽出

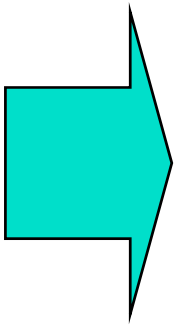
通信発生箇所の抽出

- CPY/SCLマークは消滅

8. HPFプログラミング演習(2)

- 76行目のループは、シフト通信(SFT)が発生している境界処理ループ
  - 配列aの分散次元の寸法はn=2047なので、例えば16プロセッサで並列実行する場合、a(:,1)とa(:,2)、およびa(:,n)とa(:,n-1)は同一のプロセッサ上にある
    - ON HOME指示構文+LOCAL節を指定すれば通信の抑制が可能
    - 境界処理で頻出のパタン

```
・並列化情報リスト
                                  double precision a(n,n)
                        !hpf$  distribute (*,block) :: a
                                          :
        COMM:  SFT [a] [LINO:  84  in sample.F]
        COMM:  SFT [a] [LINO:  84  in sample.F]
(    76) <N>-----------          do i=1,n
(    77)   |                       a(i,1) = a(i,2)
(    78)   |                       a(i,n) = a(i,n-1)
(    79)   +-------------        enddo
```



```
        double precision a(n,n)
!hpf$  distribute (*,block) :: a
                :
        do i=1,n
!hpf$     on home(a(:,1)), local begin
          a(i,1) = a(i,2)
!hpf$     endon
!hpf$     on home(a(:,n)), local begin
          a(i,n) = a(i,n-1)
!hpf$     endon
        enddo
```

8. HPFプログラミング演習(2)

- 再度、-Mftn、–Mlist2、–Minfoオプション付きで翻訳する
  - 並列化情報リストをチェックする

```
%> sxhpf  –Mftn  –Mlist2  –Minfo  sample.F
                  :
%> grep  “<S>”  sample.lst
(    35) <S>----------      do iter=1,maxiter
(    55) <S>----------           do i=1,n
%> grep “<P>”  sample.lst
(    38) <P>----------           do j=2,n-1
(    47) <P>----------           do i=1,n
(    51) <P>----------           do j=1,n
(    65) <P>----------         do  j=1,n
%> grep  COMM:  sample.lst
        COMM:  SFT [b]  [LINO:  38  in sample.F]
        COMM:  RED [bp]  [LINO:  51  in sample.F]
        COMM:  RED [sum]  [LINO:  65  in sample.F]
```

並列化できないと判定されたループの抽出

時間発展ループ。並列化できない。

同じループネストの54行目の<S>が消えて、並列化(<P>)されており、重い通信も出ていないので問題なし

並列化されたループの抽出

通信発生箇所の抽出
- CPY/SCLマークは消滅

76行目(境界処理部分)のSFT(シフト通信)が消滅

8. HPFプログラミング演習(2)

HPFプログラムの実行

- 実行用スクリプトをrun2.shとして用意している(4プロセスでの実行).

  % qsub run2.sh

- 実行結果を確認する. 実行結果はstep-4.oXXXX(XXXXはジョブID)として格納される.

  % cat step-4.oXXXX

9. HPF/SX V2 Rev.2.7.3 の新機能

- データ分散指定オプション
- HPFソース生成オプション
- 分散並列化情報リスト強化

データ分散指定オプション(1)

データ分散を指定できる翻訳時オプション

書式 -Mautodist[=rank*?*[:*n*]]

- 既定値では，全配列を最後の次元でBLOCK分散する (分散指定のない配列のみ)
- =rank? を指定した場合，?次元配列を最後の次元でBLOCK分散する
- =rank?:n を指定した場合，2進整数nの1に対応する次元をBLOCK分散する

- **いろいろなデータ分散を気軽に試せる**
- **規則的なデータ構造のコードなら，オプションのみで並列化できる場合も**

注記)

- DISTRIBUTE**指示文**，ALIGN**指示文**，INHERIT**指示文**，DYNAMIC**指示文，又は** SEQUENCE**指示文が記述されている配列は，指示文による指定が優先されます。**
- **共通ブロックや**interface block**中に配列が出現する場合，全ての手続に同一のデータ分散指定オプションを指定する必要があります。**
- COMMON**文と**NAMELIST**文の両方に出現する配列がある場合，本オプションは使用できません。**
- **次のような配列は分散されません。**
  - PARAMETER**文，**EQUIVALENCE**文，又は** NAMELIST**文中に出現する配列**
  - **大きさ引継ぎ配列，文字型の配列，及び 派生型の配列**
  - POINTER**属性 又は** TARGET**属性を持つ配列**
  - LOCAL**手続中の仮引数以外の配列**

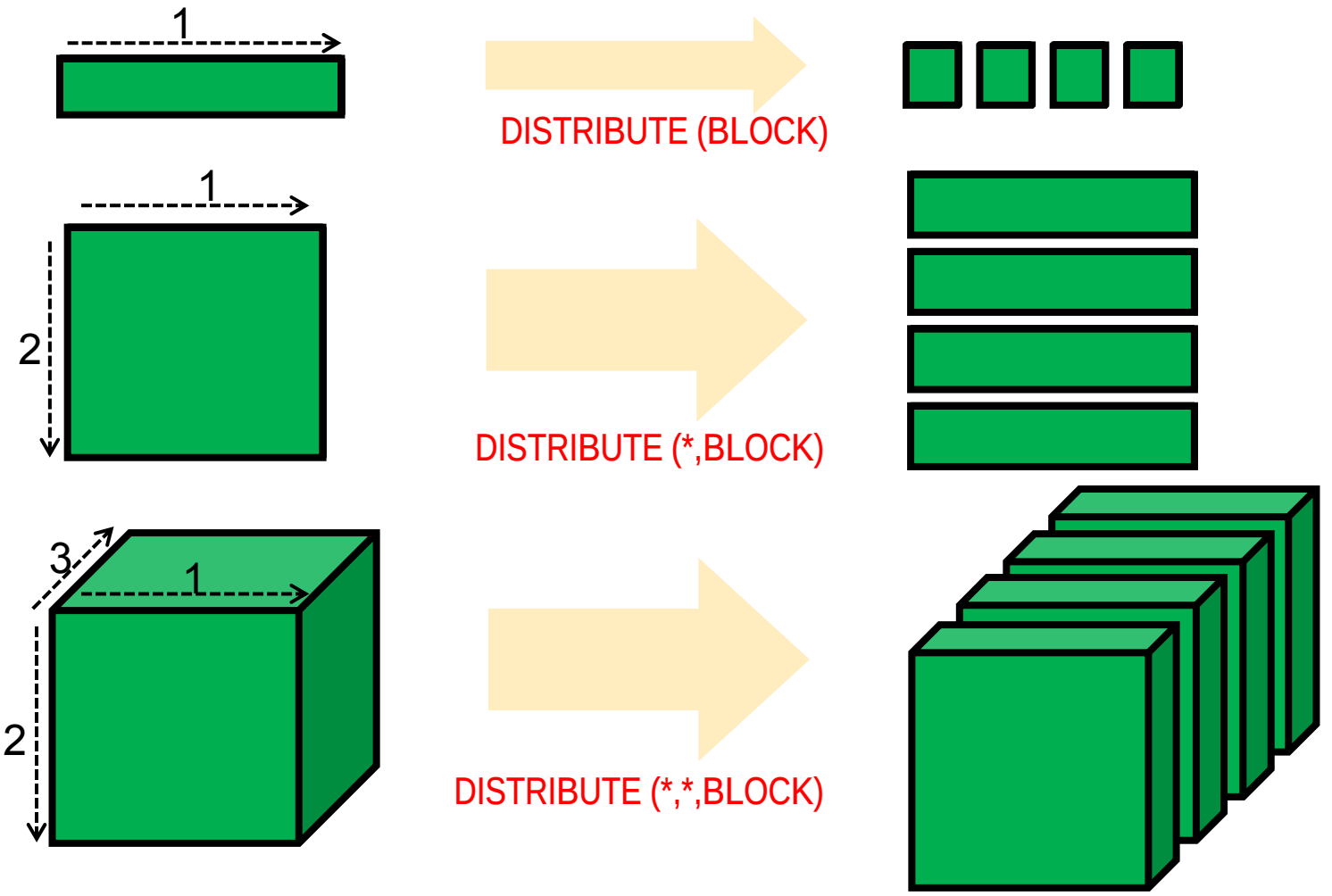
データ分散指定オプション(2)

例1)

全ての配列の最後の次元をBLOCK分散する場合

(**指定される分散は，**-Minform=inform**オプションで確認できます**)

%> sxhpf -Mautodist sample.hpf

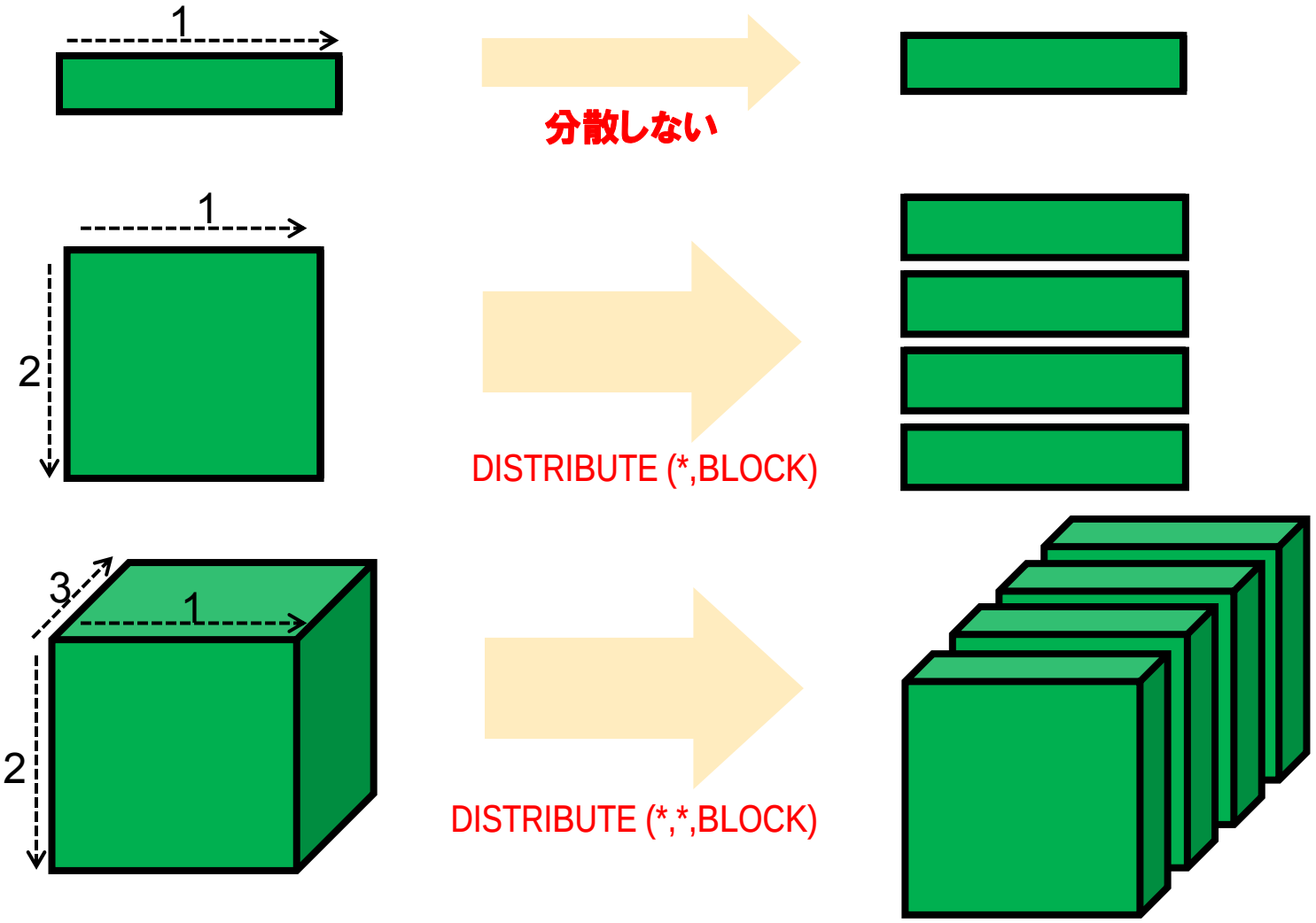


データ分散指定オプション(3)

例2)

2次元配列と3次元配列の最後の次元をBLOCK分散する場合

%> sxhpf –Mautodist=rank2 -Mautodist=rank3 sample.hpf

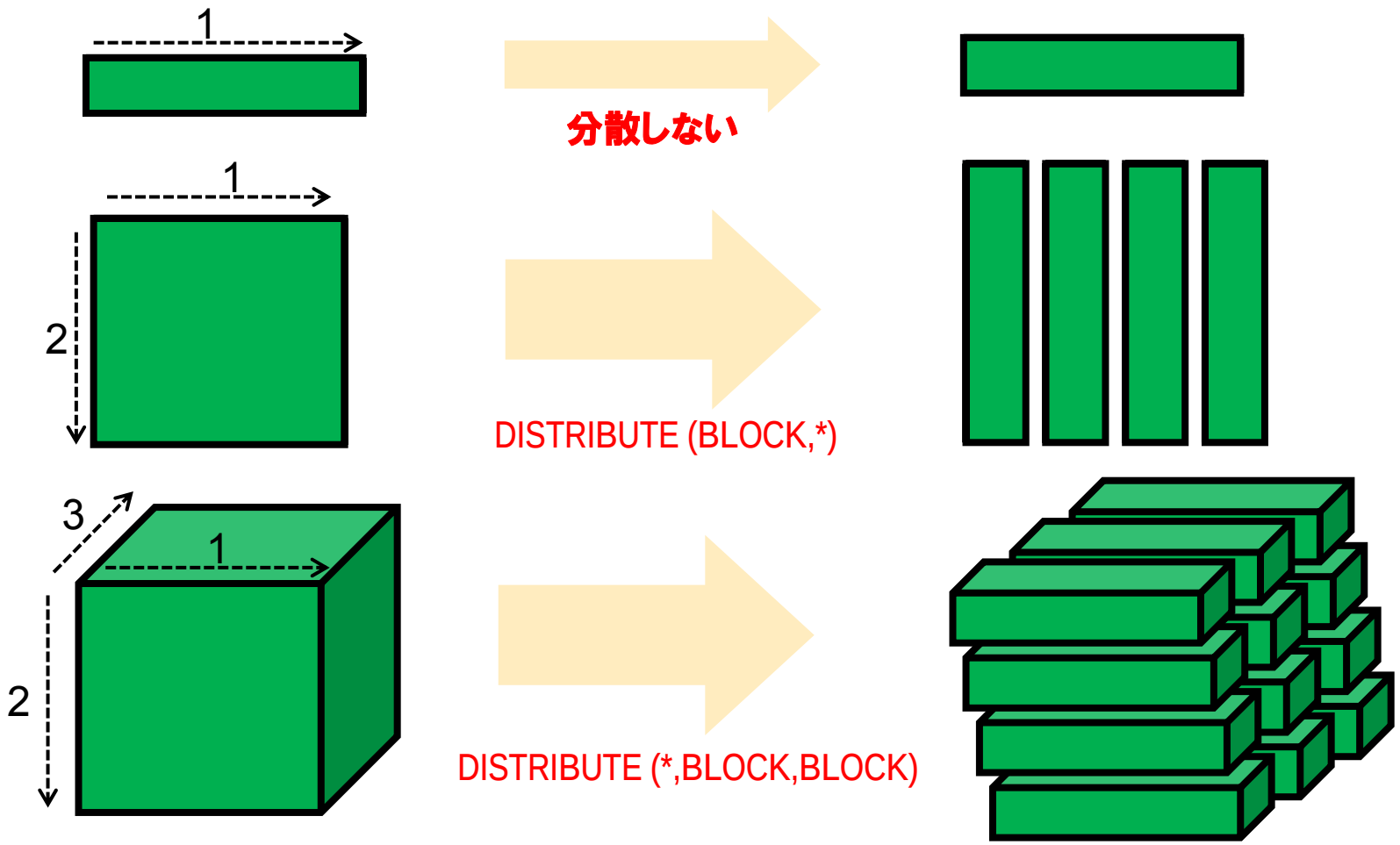


データ分散指定オプション(4)

例3)

2次元配列の1次元目 及び 3次元配列の2次元目・3次元目を BLOCK分散する場合

%> sxhpf –Mautodist=rank2:10 -Mautodist=rank3:011 sample.hpf



データ分散指定オプション(5)

使用方法の例

1. データ分散指定オプションで，主要配列を最後の次元で分散してみる。
   - 同時に，-Mlist2 オプションを指定し，分散並列化情報リストを出力する
2. 分散並列化情報リストの COMM: マークをgrepし，通信が多発していないかチェックする
   - 特に 一要素毎の通信 (SCL) や 配列コピー (CPY) パタンは要注意
   - 必要に応じて，HPF指示文の追加を行う。例えば
     - 通信が，並列化できないと判定されたループ (<S> マーク)で発生しており，本当は並列化可能なループであるなら，INDEPENDENT指示文(+REDUCTION節)を追加してみる。
     - 或いは，境界処理のループなら，ON HOME指示文 + LOCAL節を追加して，両端のプロセッサだけで実行すれば，通信が不要であることを指示してみる。
3. COMM: マークが少なくなるまで，指定する分散の変更や，ソースへの指示文の追加を試してみる。

HPFソース生成オプション(1)

HPFソースプログラムを自動生成する翻訳時オプション

書式 -Mhpfout

- -Mautodistオプションで指定したデータ分散入りのソースを出力する

- **出力されたソースをベースに，チューニングを行うことができる**
- **データ分散の指定し忘れ などのミス防止にも有効**

注記) INCLUDLE**行は展開された形で出力されます**

HPFソース生成オプション(2)

例）

配列の最後の次元を分割するHPF指示文を挿入したソースを生成

```
%> sxhpf   -Mautodist   -Mhpfout   sample.hpf
```

```
      real, dimension(100,100)  :: a,b,c
!hpf$ distribute (*,*) :: a  ! 非分散
      do j=1,100
        do k=1,100
          do i=1,100
            c(i,j) = c(i,j) + a(i,k)*b(k,j)
          enddo
        enddo
      enddo
```

sample.hpf

```
      real, dimension(100,100)  :: a,b,c
!hpf$ distribute a(*,*)  ! 非分散
!hpf$ distribute b(*,block)
!hpf$ distribute c(*,block)
      do j=1,100
        do k=1,100
          do i=1,100
            c(i,j) = c(i,j) + a(i,k)*b(k,j)
          enddo
        enddo
      enddo
```

sample.hpf.src

分散並列化情報リスト強化

自動共有並列化・ベクトル化情報を分散並列化情報リストに出力

●FORTRAN90/SXの編集リスト出力オプション(-R2, -R5, 又は –Wf"-fmtlist")と，分散並列化情報リスト出力オプション(-list2)を同時に指定すると，分散並列化情報リスト中に，自動共有並列化，ベクトル化情報を出力する。

注記) **配列構文に対しては，マークは出力されません**

例) %> sxhpf -Mlist2 –R5 –Pauto **オプション指定時の出力例**

ベクトル化も並列化もされなかった

ベクトル化のみ

分散並列化+自動共有並列化

ベクトル化のみ

```
(    5)  <S>+----------------------          do i=1,99
         COMM:  SFT  [a]  [LINO:     5  in src.hpf]
(    6)  <N>V-----------------------            do j=1,100
(    7)    |                                       a(j,i) = a(j,i) + a(j,i-1)
(    8)    +------------------------            enddo
(    9)                                       enddo
         COMM:  RED  [x]  [LINE:   10  in src.hpf]
         HOME:  a(:,j)
(   10)  <P>P----------------------          do j = 1,100
(   11)    | <I>V------------------            do i = 1, 100
(   12)    |                                     x = max(x,a(i,j))
(   13)    |                                   enddo
(   14)    +------------------------          enddo
```

10. HPFプログラミングのテキスト紹介

http://www.hpfpc.org/book/hpf_text.html

培風館 2011年3月30日発売
ISBN978-4-563-01586-2 ¥3400+税

HPFによる並列プログラミング手法唯一の実践的な解説書。(文法書に類するものはあった)

High Performance Fortran2.0公式マニュアル
(シュプリンガー・フェアラーク東京)
The High Performance Fortran Handbook
(The MIT Press)

対象読者

- (分散)並列計算に興味のある初心者から上級者まで
- **初歩的な**Fortranの知識は仮定
(DO文, 代入文, IF文程度)

著者

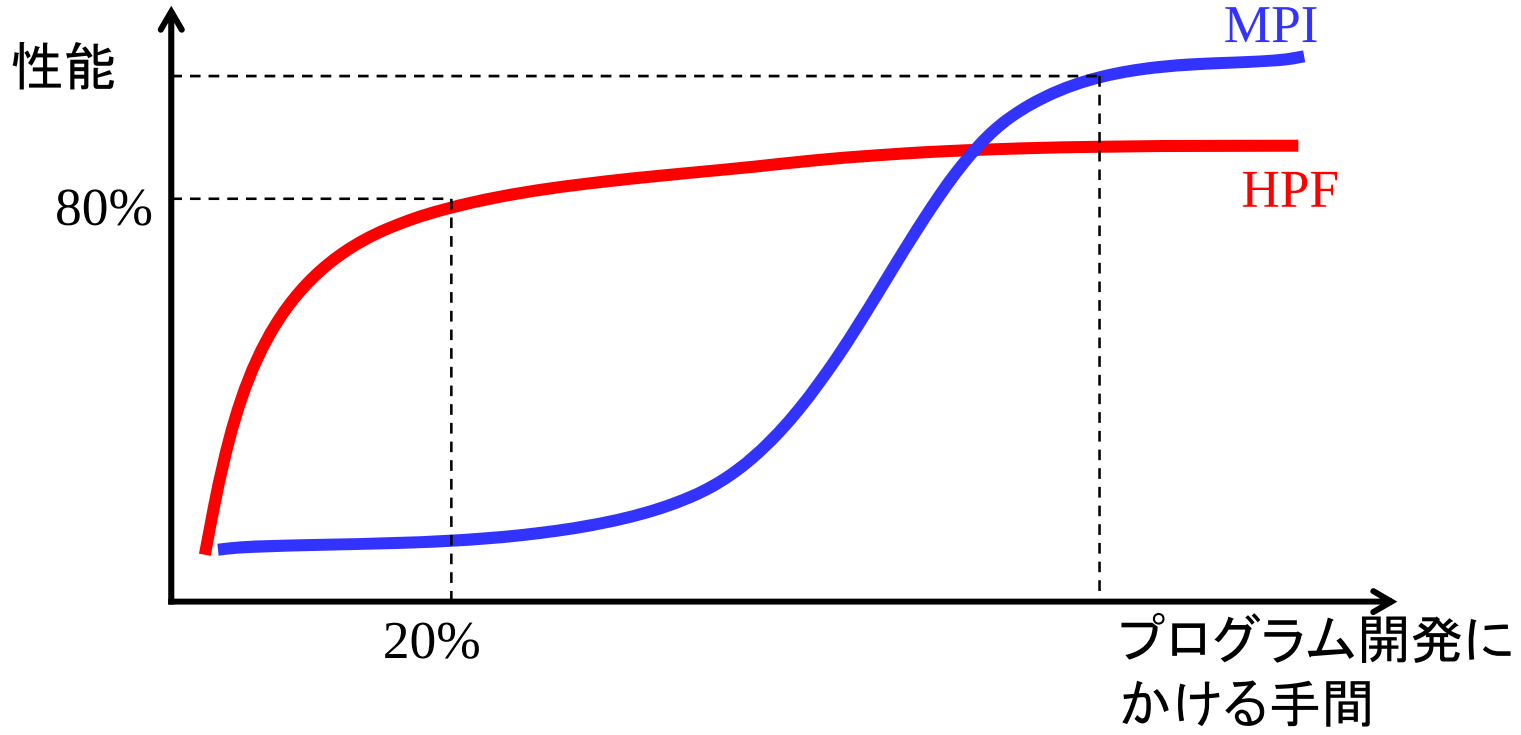
監修　津田孝夫
編集　HPF推進協議会
著者　岩下英俊・坂上仁志・妹尾義樹・林康晴

さいごに

「MPIの 20%の手間 で 80%の性能」

プログラム開発・保守を効率化

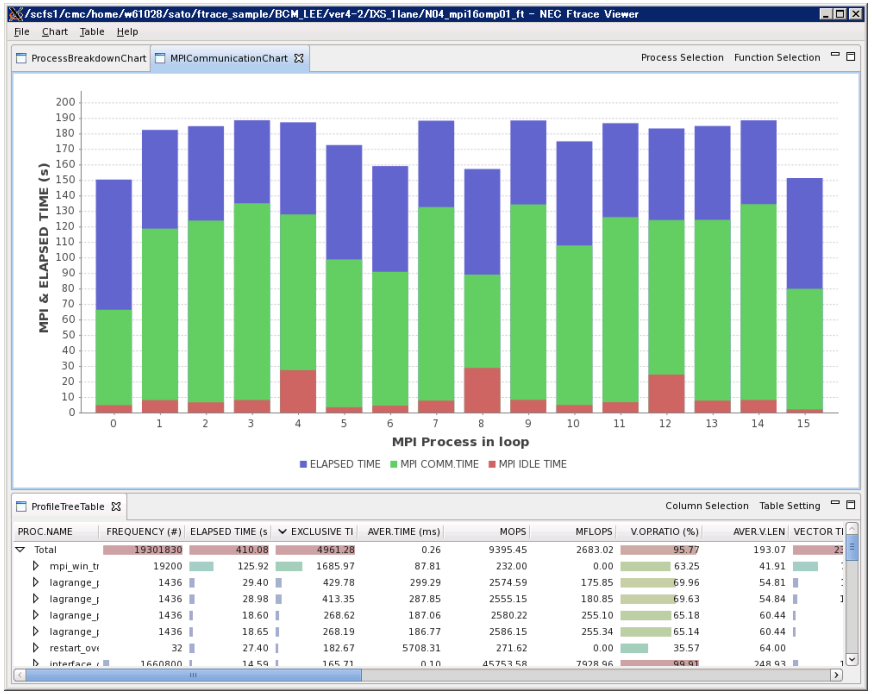
本来の研究に注力



【補足資料】NEC Ftrace Viewer

簡易性能解析機能(ftrace)情報をグラフィカルに表示するためのツール

- 関数・ルーチン単位の性能情報を絞り込み、多彩なグラフ形式で表示できます.
- 自動並列化機能・OpenMP・MPIを利用したプログラムのスレッド・プロセス毎の性能情報を容易に把握できます.



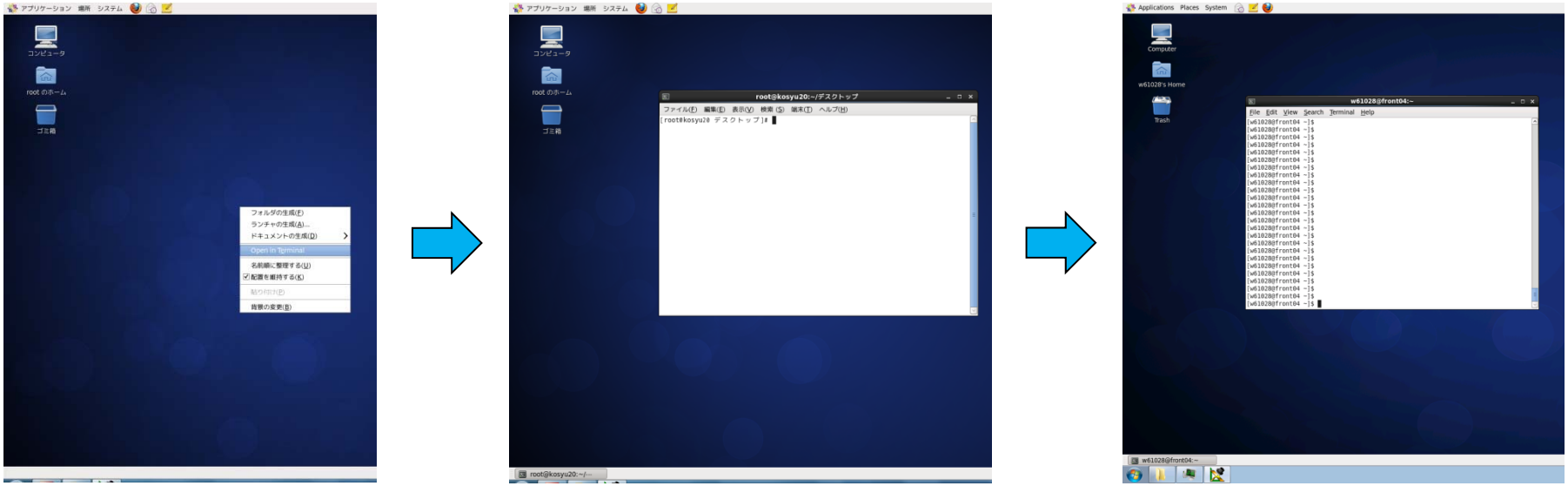
【補足資料】NEC Ftrace Viewer

NEC Ftrace Viewerの使用方法について

1．環境（Xサーバ）の準備（Exceedの場合）

フロントエンドマシンへのログイン

- Exceedの起動
- 端末画面の起動
  - マウス右クリックで表示されるメニューから “Open in Terminal” を選択
- フロントエンドマシンへログイン



1．環境（Xサーバ）の準備（Xmingの場合）

フロントエンドマシンへのログイン

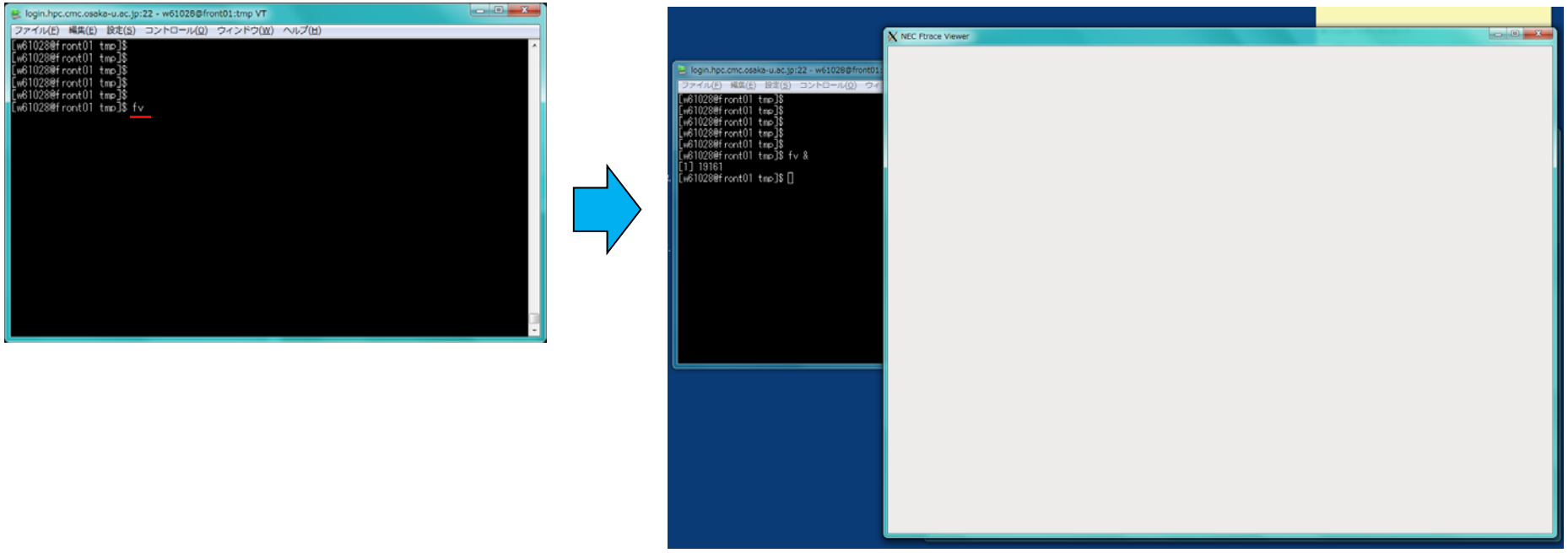
- Xmingの起動
  - 「すべてのプログラム」→「Xming」→「Xming」でXmingを起動．
  - Windows環境では，起動するとタスクバーにXmingのアイコンが表示される．
- TeraTermの設定
  - 「設定」→「SSH転送」→「リモートの（X）アプリケーション…」のチェックを入れてOKを押下．
  - xeyesコマンドなどで，画面転送ができているか確認して下さい．
- フロントエンドマシンへログイン

※次ページからの説明は<u>Windows環境でXmingを使用した場合</u>を例にしています．

2. NEC Ftrace Viewer の起動

GUI 画面の表示

- “fv”コマンドの実行
- Xmimgウィンドウが立ち上がり，NEC Ftrace Viewer画面が表示される．



3. ファイルの読み込み

初期画面の上部メニュー「File」から表示する ftrace.out を選択

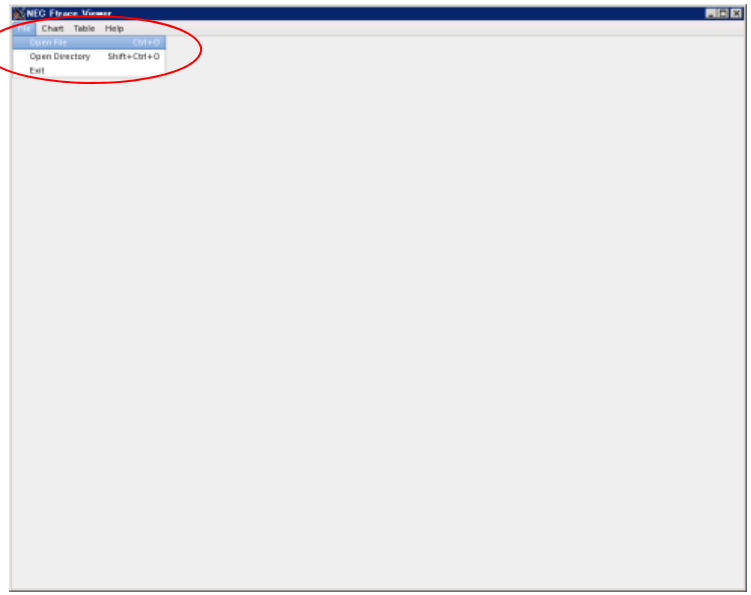
- Open File
  - 指定した ftrace.out もしくは ftrace.out.n.nn を1つ読み込みます.
- Open Directory
  - 指定したディレクトリ直下の ftrace.out もしくは ftrace.out.n.nn を全て読み込みます.

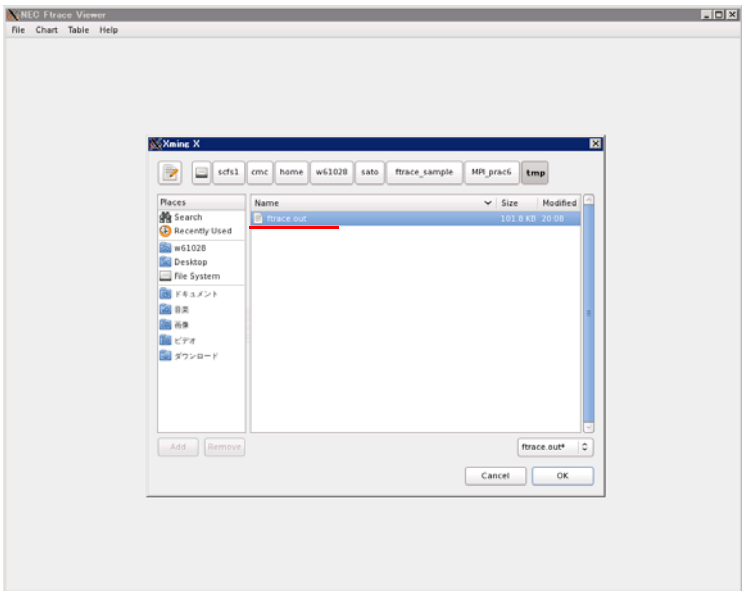
  ※ ftrace.out と ftrace.out.n.nn が同じディレクトリにある場合、読み込みに失敗します.

3. ファイルの読み込み

シリアル/SMP実行の場合(1/2)

- ftrace.out ファイルの読み込み
  - 「File」→「Open File」から読み込みたい ftrace.out を選択して「OK」を押下

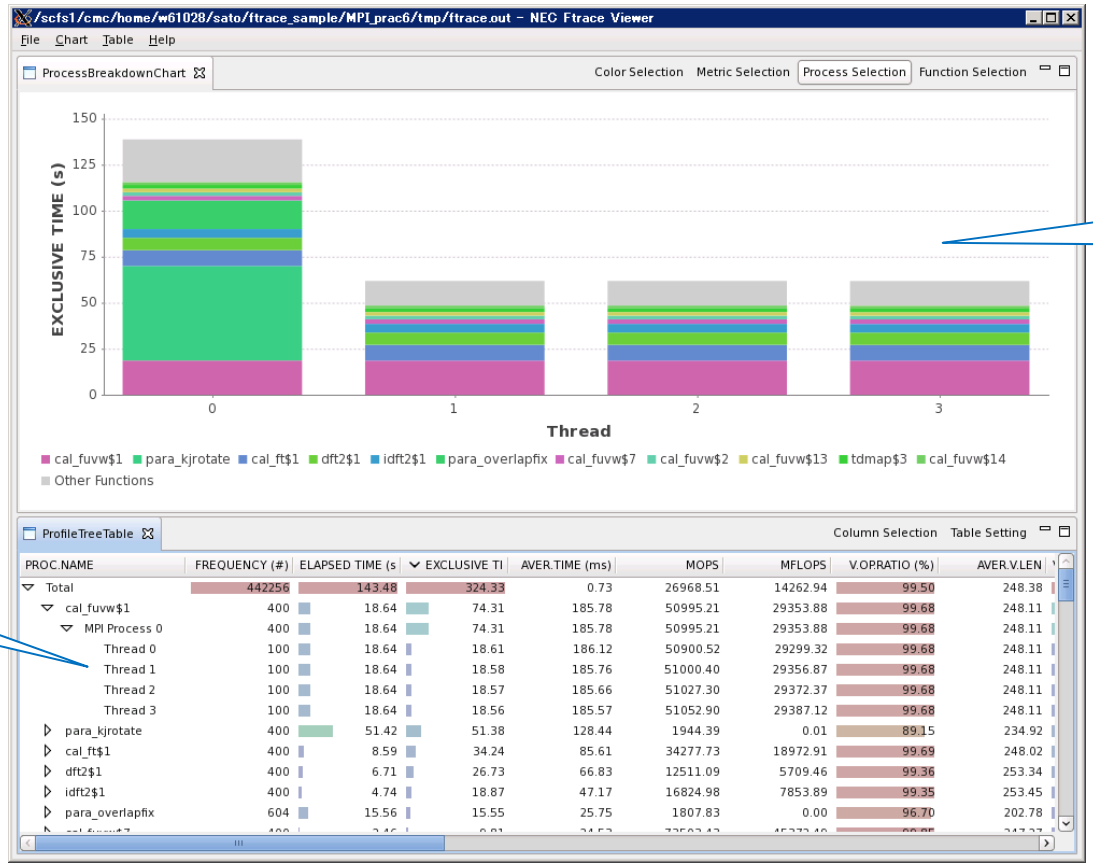




3. ファイルの読み込み

シリアル/SMP実行の場合(2/2)

- GUI画面の例(4SMP実行の結果)
  - “Process Breakdown Chart”モード



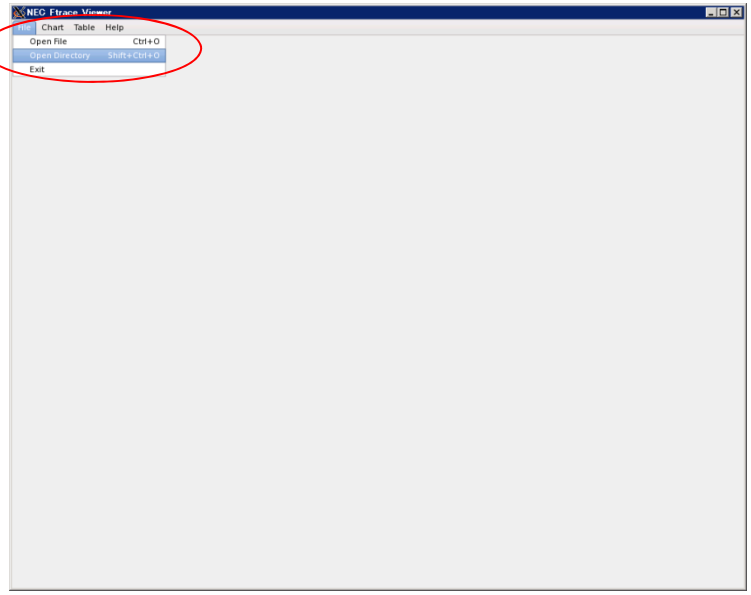
右クリックでグラフを
画像として保存できます

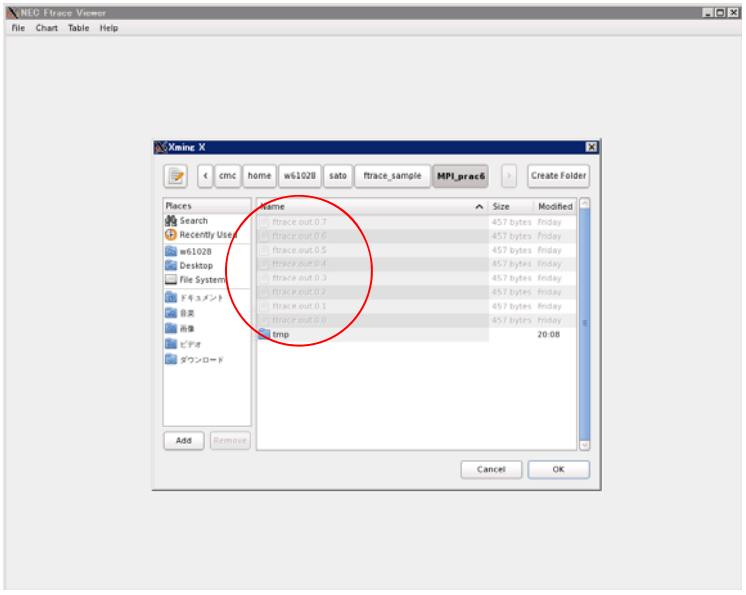
SMP並列プロセスごとの
性能情報が表示されます

3. ファイルの読み込み

MPI実行の場合(1/2)

- ftrace.out.n.nn ファイルの読み込み
  - 「File」→「Open File」から読み込みたい ftrace.out.n.nn があるフォルダを選択して「OK」を押下
    - ✓ 今回は，MPIプロセス分の ftrace.out ファイルを読み込む場合を例にしています．
    - ✓ 1プロセス分だけを表示させる場合は，「シリアル/SMP実行の場合」のようにプロセスに対応した ftrace.out.n.nn を指定して下さい．

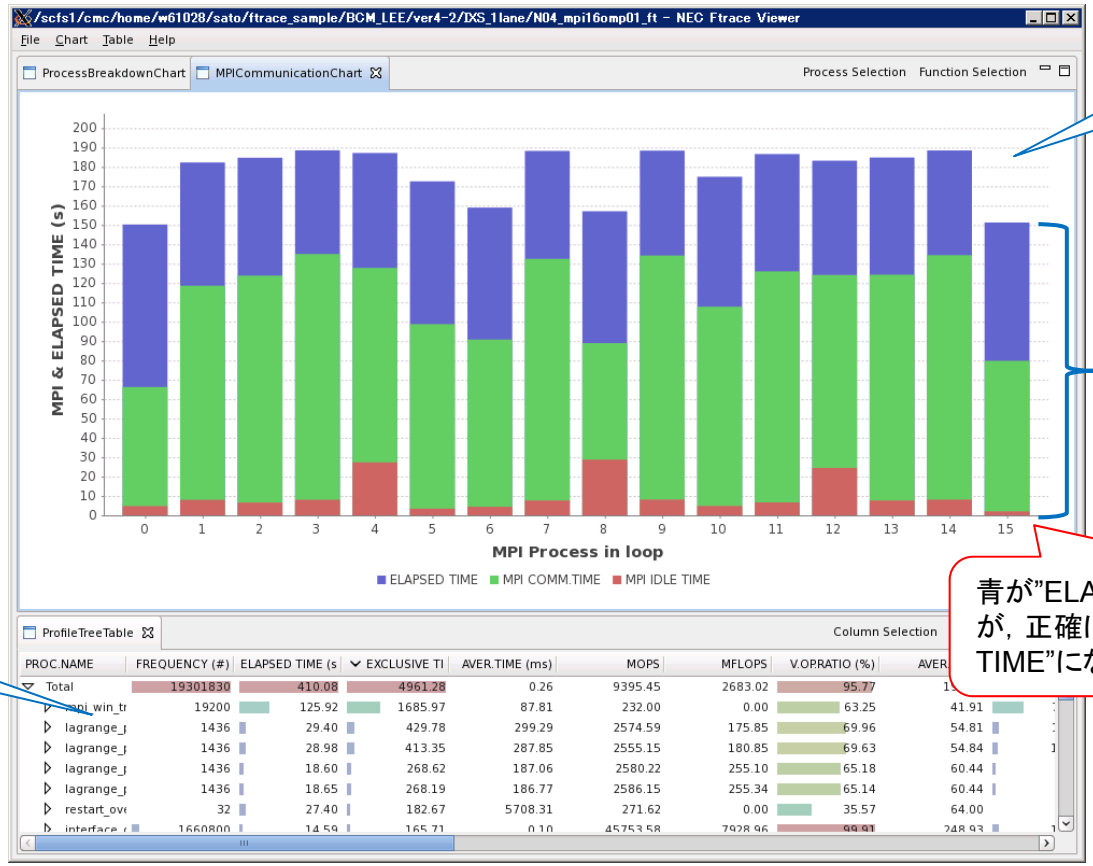




3. ファイルの読み込み

MPI実行の場合(2/2)

- GUI画面の例(16MPI実行の結果)
  - “MPI Communication Chart”モード



右クリックでグラフを
画像として保存できます

ELAPSED TIME

MPI並列プロセスごとの
性能情報が表示されます

青が”ELAPSED TIME”になっていますが，正確には青＋緑+赤が”ELAPSED TIME”になります．

Empowered by Innovation

NEC