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

NEC \Orchestrating a brighter world

H27年度
新スーパーコンピュータにおける
高速化技法の基礎

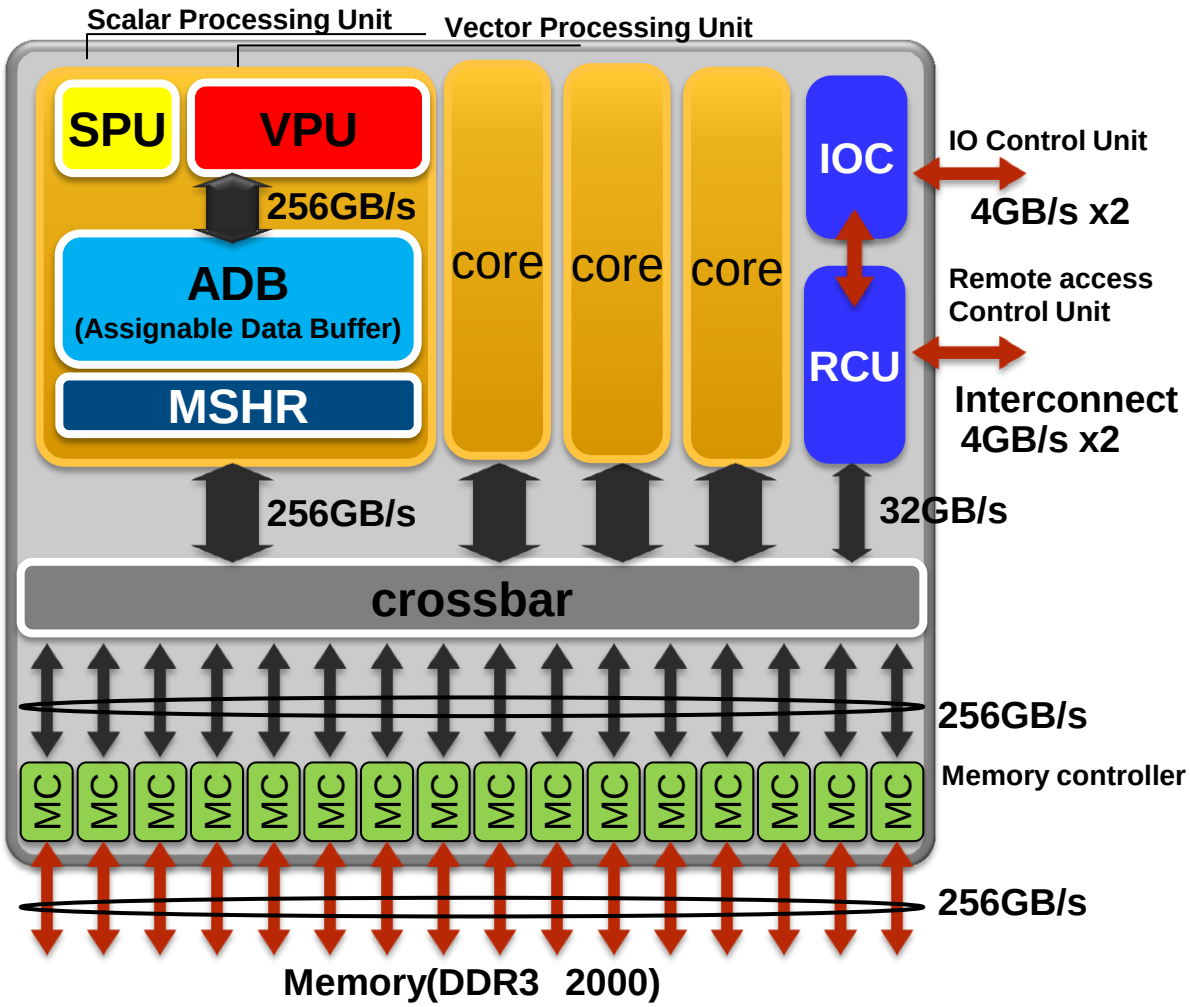
2015年　7月31日

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. SX-ACE概要

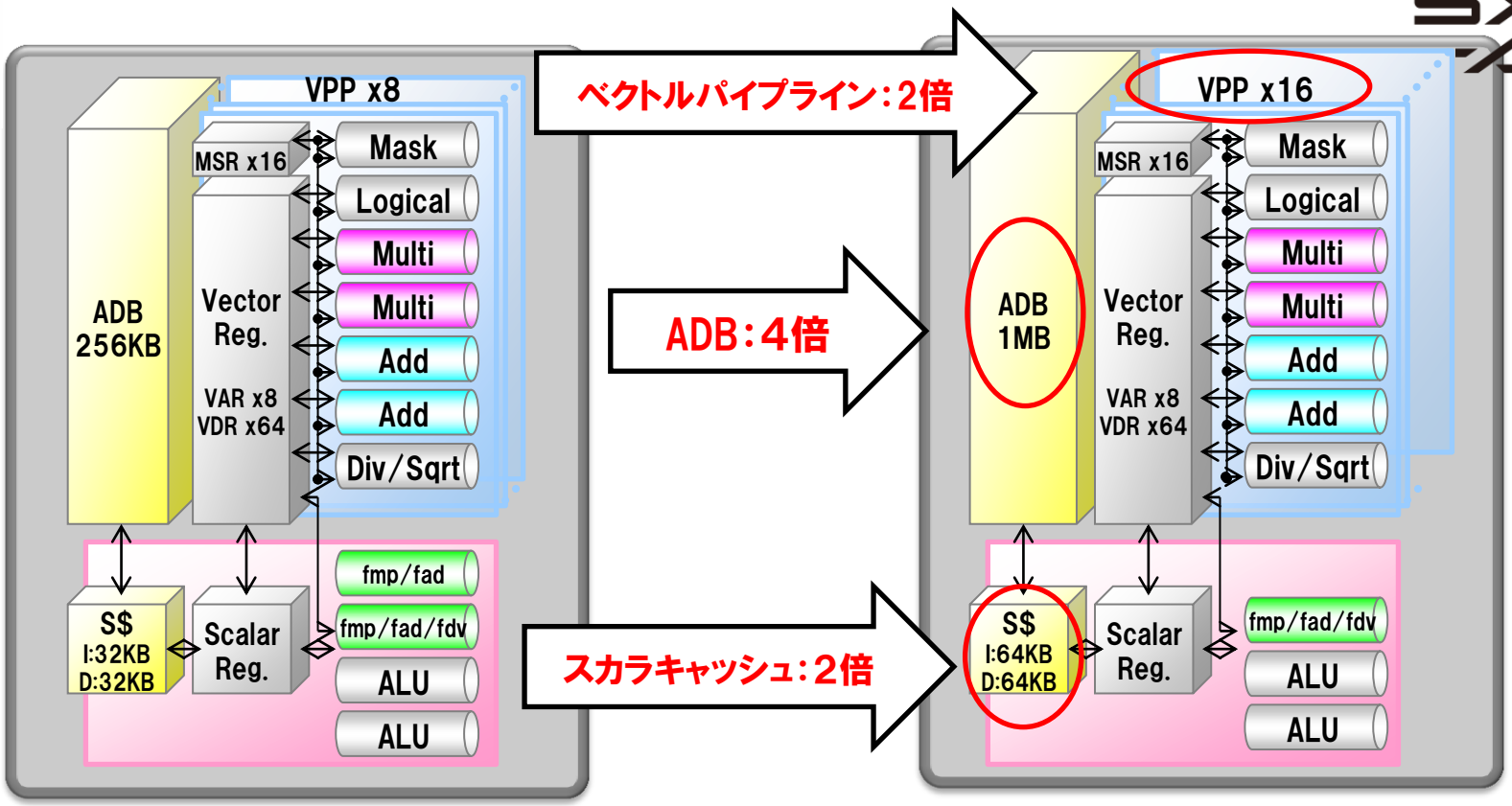
CPUアーキテクチャ



アーキテクチャ	
コア	
演算性能	64GFlops
ADBサイズ	1MB
ADB帯域	256GB/s
メモリ帯域	64～256GB/s
CPU	
コア数	4
演算性能	256GFlops
メモリ帯域	256GB/s
Byte/Flop	1

(※MSHR：Miss Status Handling Registers)

演算器


VPP x8
MSR x16
Mask
Logical
Multi
Multi
Add
Add
Div/Sqrt
ADB
256KB
Vector
Reg.
VAR x8
VDR x64
S$
I:32KB
D:32KB
Scalar
Reg.
fmp/fad
fmp/fad/fdv
ALU
ALU
ベクトルパイプライン：2倍
ADB：4倍
スカラキャッシュ：2倍
VPP x16
MSR x16
Mask
Logical
Multi
Multi
Add
Add
Div/Sqrt
ADB
1MB
Vector
Reg.
VAR x8
VDR x64
S$
I:64KB
D:64KB
Scalar
Reg.
fmp/fad/fdv
ALU
ALU

SX-ACEの強化ポイント(対SX-9)

ベクトル性能強化

短ベクトル性能強化
- ■SPUベクトルパイプライン新設
- ■演算器間のダイレクトバイパスチェイニング

リストベクトル性能強化
- ■メモリレイテンシ短縮
- ■命令追い越し実行機能強化

スカラ性能強化

パイプライン構成見直し
L1キャッシュ容量拡張(2倍化)
分岐予測機能強化
命令発行機能強化
ハードウェアプリフェッチ機能

メモリ性能強化

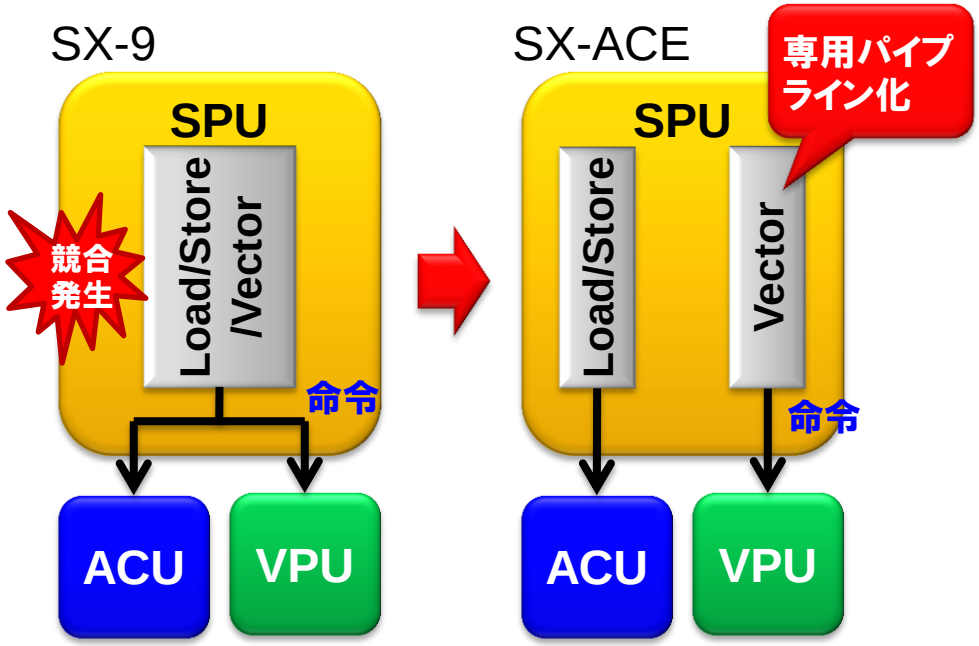
ADB容量拡張(1MB/コア)
MSHRによる冗長なロードメモリアクセス数削減
ストア圧縮による冗長なストアメモリアクセス数削減

短ベクトル性能強化(対SX-9)

命令発行強化，演算立上り時間短縮により，短ベクトル性能を強化

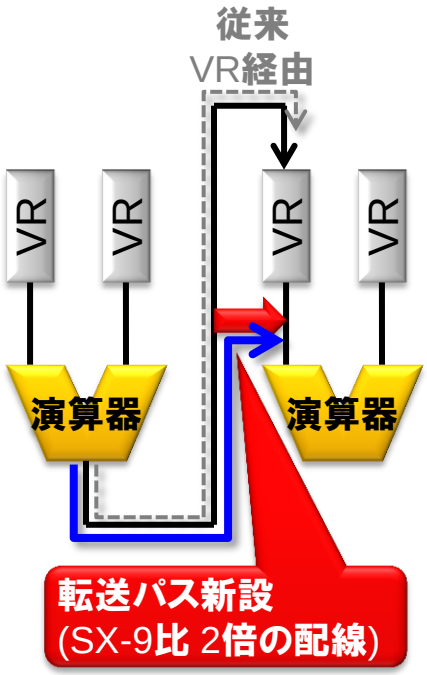
SPUベクトルパイプライン追加

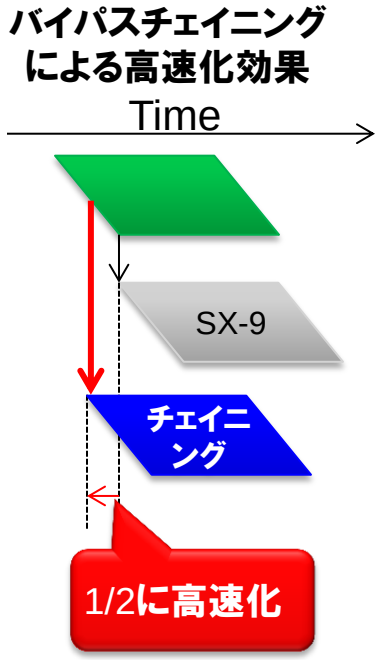
ベクトル命令発行性能を大幅強化し，
短ベクトル時のVPU命令枯渇を防止



バイパスチェイニング機能

演算器間のダイレクトなデータ転送を可能
とし，連続する演算の待ち時間を短縮





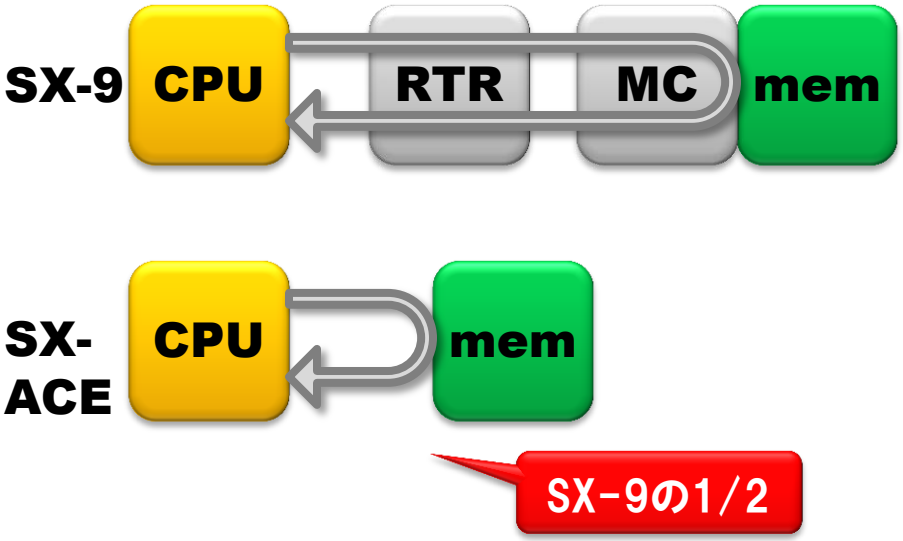
(ACU： Access Control Unit VR : Vector Resister)

リストベクトル性能強化(対SX-9)

**メモリレイテンシ短縮，命令間追い越し機能強化により
リストベクトル性能を強化**

メモリレイテンシ短縮

CPU/DRAM直結のアーキテクチャ採用により，メモリアクセスレイテンシを1/2に短縮



(RTR：ルーターモジュール)

VOフラグによる追い越し強化

ベクトルメモリアクセス命令のVO(Vector Overtake)フラグによるメモリアクセス命令間の追越機能を強化．VOフラグはコンパイラ/ユーザ指定による利用が可能

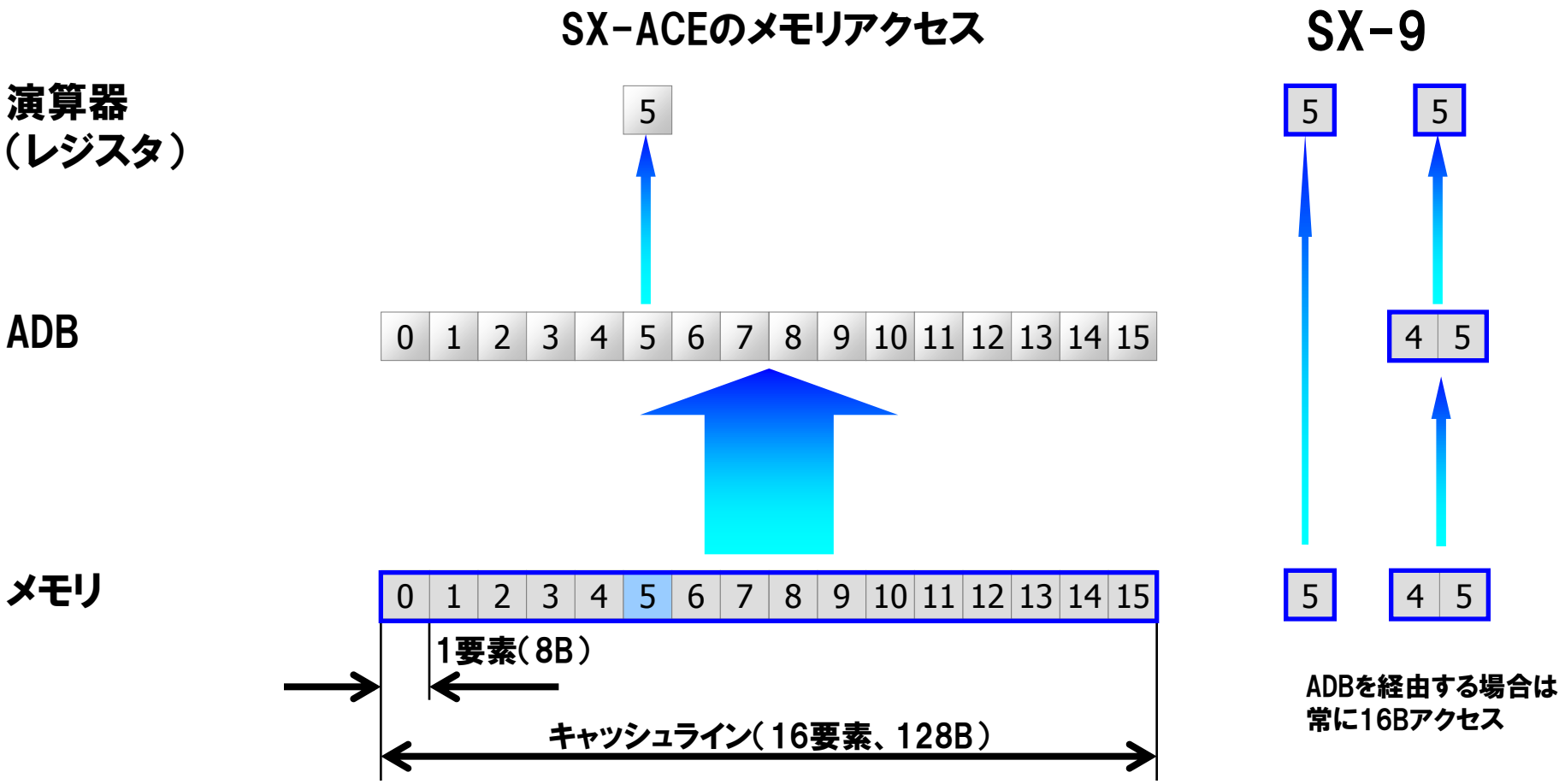
後続命令 \ 先行命令	VLD	VGT	VST	VSC
VLD	◎	◎	○	△
VGT	×	×	★	★

SX-ACE新機能

◎ アドレスによらず追い越し可能
○ アドレス競合なし時追い越し可能
★/△ 先行命令がVOフラグ設定時追い越し可能

SX-9とSX-ACEのメモリアクセス単位について(1/2)

SX-ACEはメモリアクセス単位が128B(従来SXは8B or 16B単位)

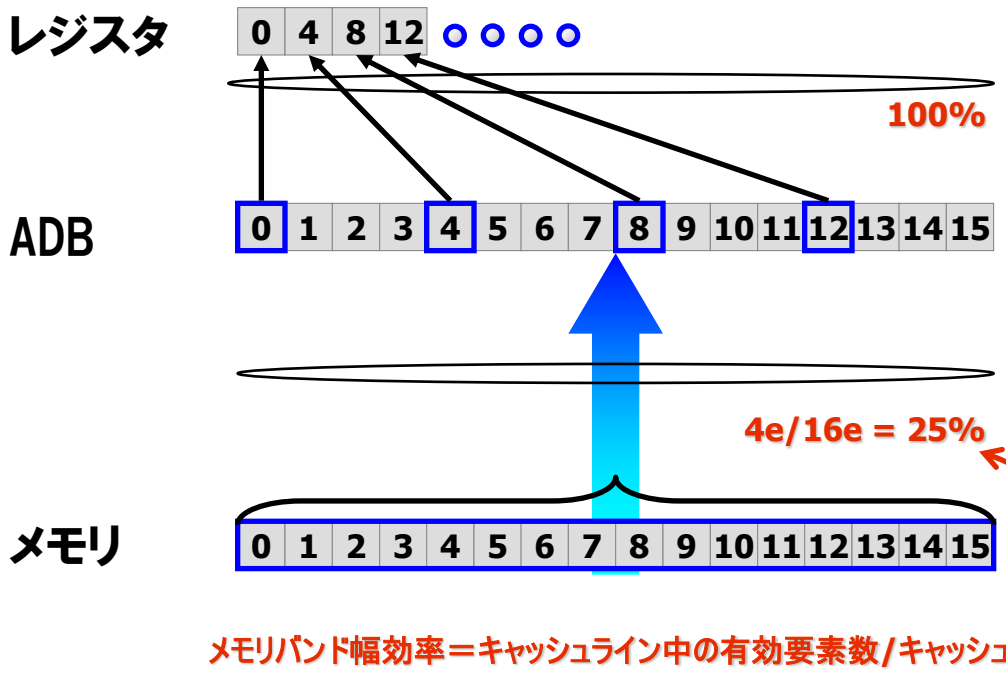


SX-9とSX-ACEのメモリアクセス単位について(2/2)

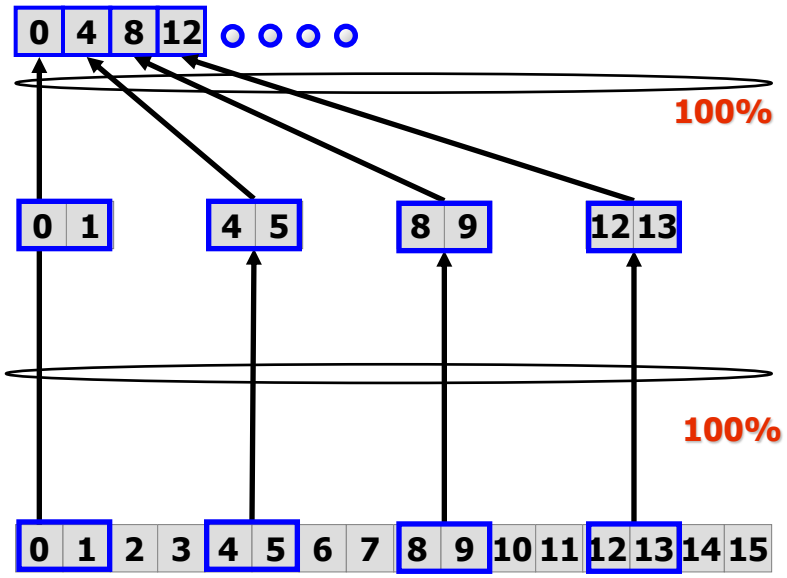
ストライドアクセスの場合(例:4要素飛びアクセス)

- **SX-ACE　: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-9　　:1要素 or 2要素単位でのメモリアクセスが可能の為、性能低下なし**

SX-ACEのメモリアクセス



SX-9のメモリアクセス



メモリバンド幅効率＝キャッシュライン中の有効要素数/キャッシュラインサイズ(16e)

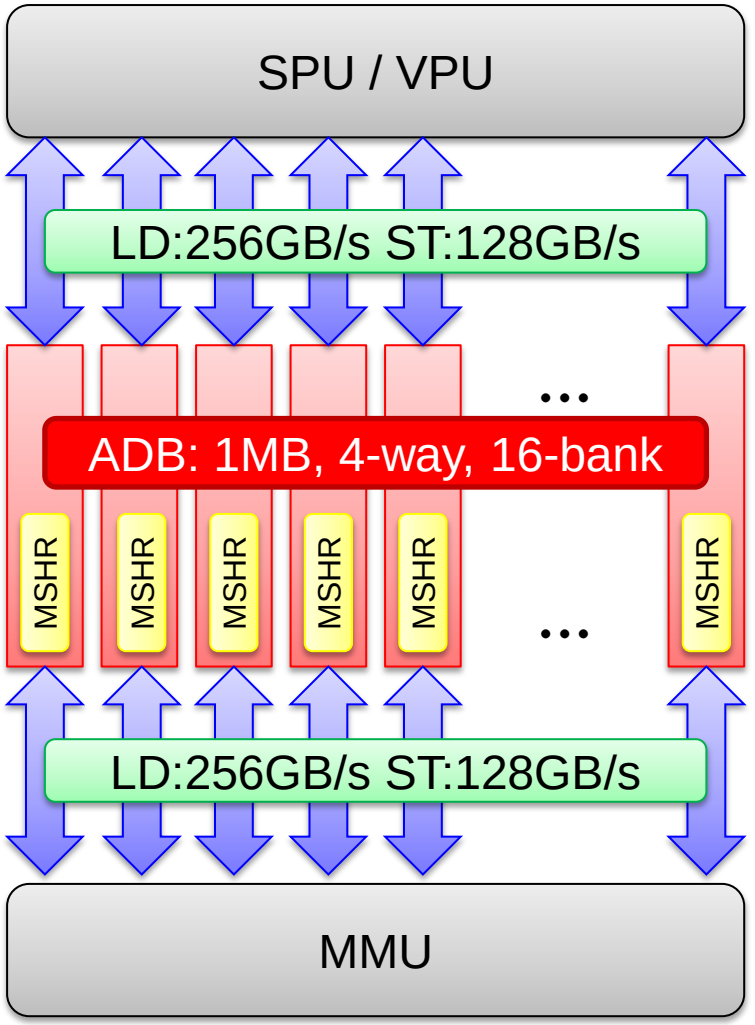
Assignable Data Buffer（ADB）

■ ADBの特徴

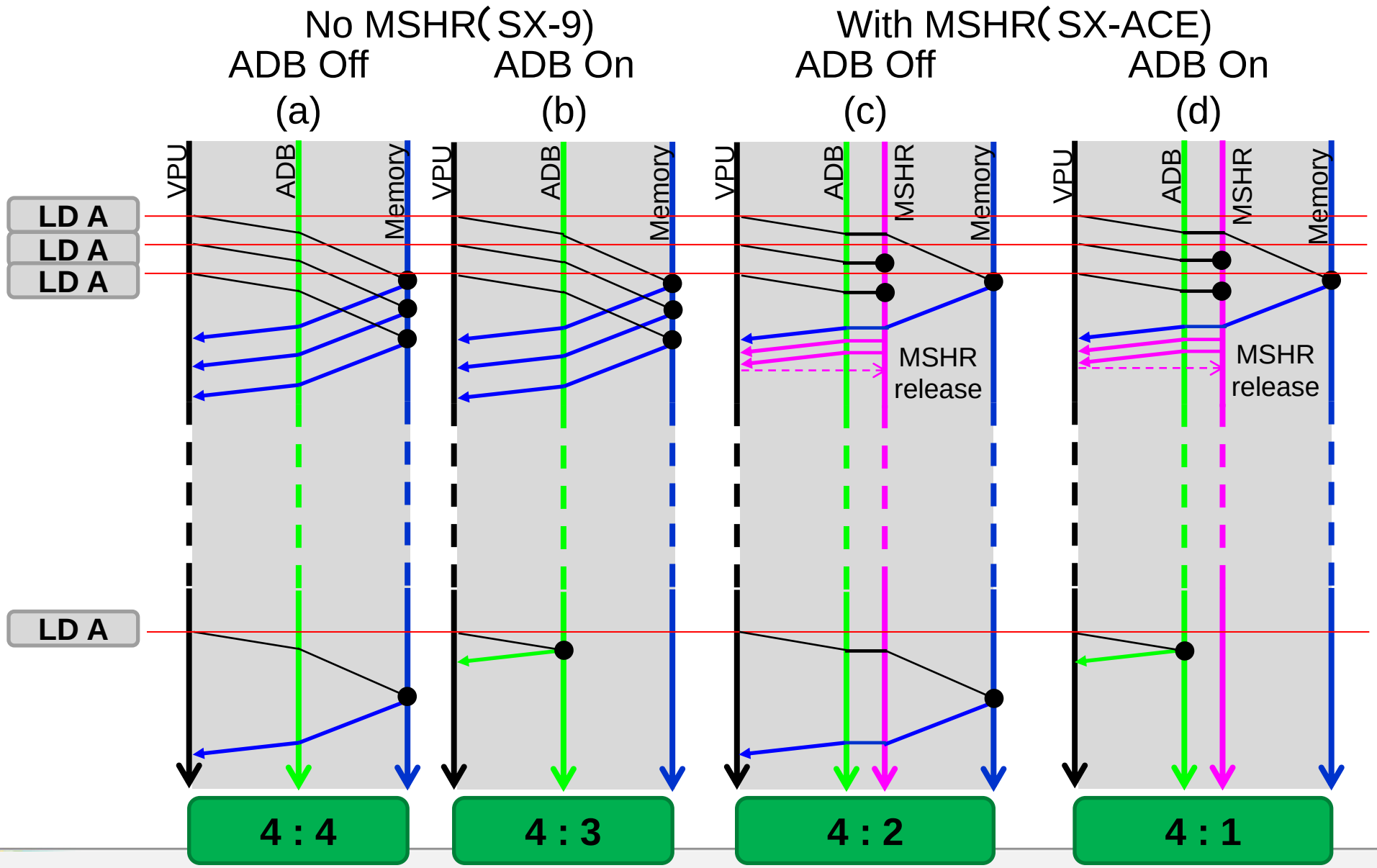
- SX-9で実装したADB機能を強化
- データの格納可否をSW制御し，高い利用効率を実現
- リストアクセスなどのメモリレイテンシが重要なメモリアクセスを高速化
- コア毎に容量1MB・帯域256GB/s（4B/F）

■ ADBの効果

- アクセスデータをADB上に保持することにより，短アクセスレイテンシ，高メモリ帯域を実現，かつメモリへのアクセス回数を削減し実効メモリ帯域を向上させる



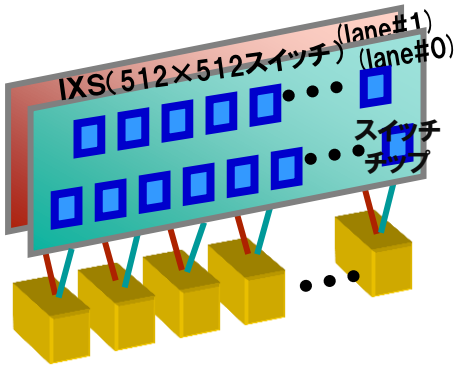
ADB/MSHRの動作概念

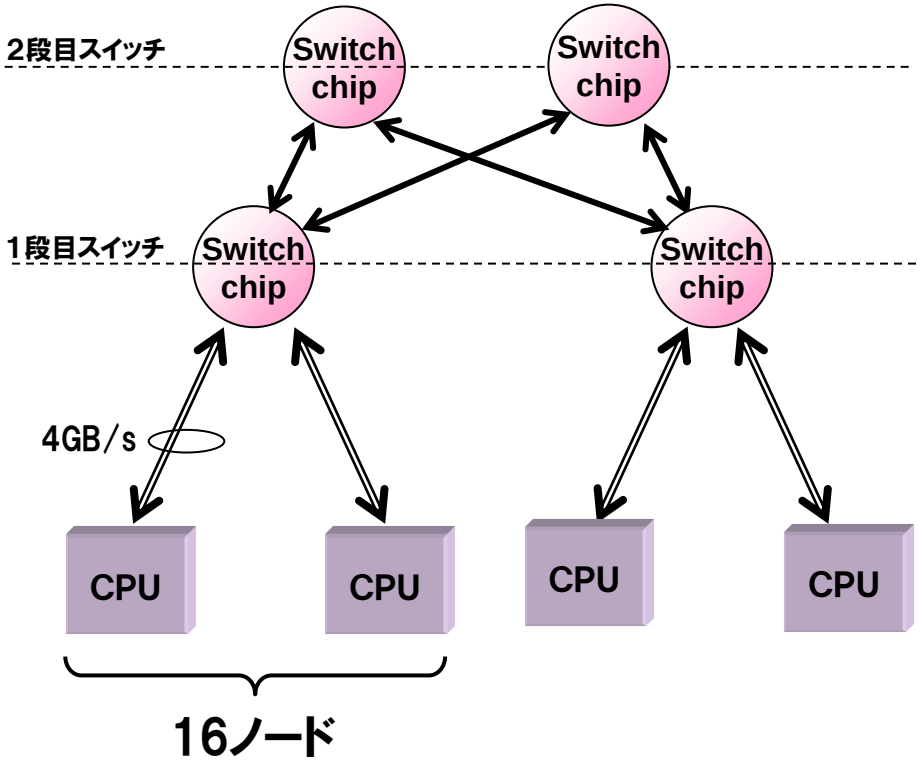


ノード間ネットワーク性能

ノード間転送性能

- 2段ファットツリー（フルバイセクション）
- 任意のノード間で，ノード当たり4GB／s
  ※SX-9はノード当たり64GB/s（CPUあたり8GB/s）





2. SX-ACE新機能

コンパイラ指示行

SX-ACE(SUPER-UX R21.1)から追加されたもしくは変更された主な指示行は以下の通り.

カテゴリ	指示行名	内　容
最適化関連	ARRAYCOMB	配列式のループ融合．他の指示オプションが指定できるように変更
	DATA_PREFETCH	ADBへプリフェッチを行う
ベクトル化関連	NOCONFLICT	メモリの重なりがないことを指定し，命令追い越しを可能にする
	GTHREORDER	リストベクトルの重なりがないことを指定し，命令並べ替えを行う
	VOVERTAKE，VOB	より大胆な命令追い越しを可能にする
デバッグ関連	TRACEBACK	実行中にトレースバックを表示する

ベクトル化関連の指示行を以降で説明(他は補足資料として添付).

GTHREORDER指示行

- **直後のループ中のリストベクトルロード・ストアは互いに重なることがないものと仮定してコンパイル時に命令の並べ換えを行う.**
- **実際には重なりがある場合，結果不正となるので注意が必要.**

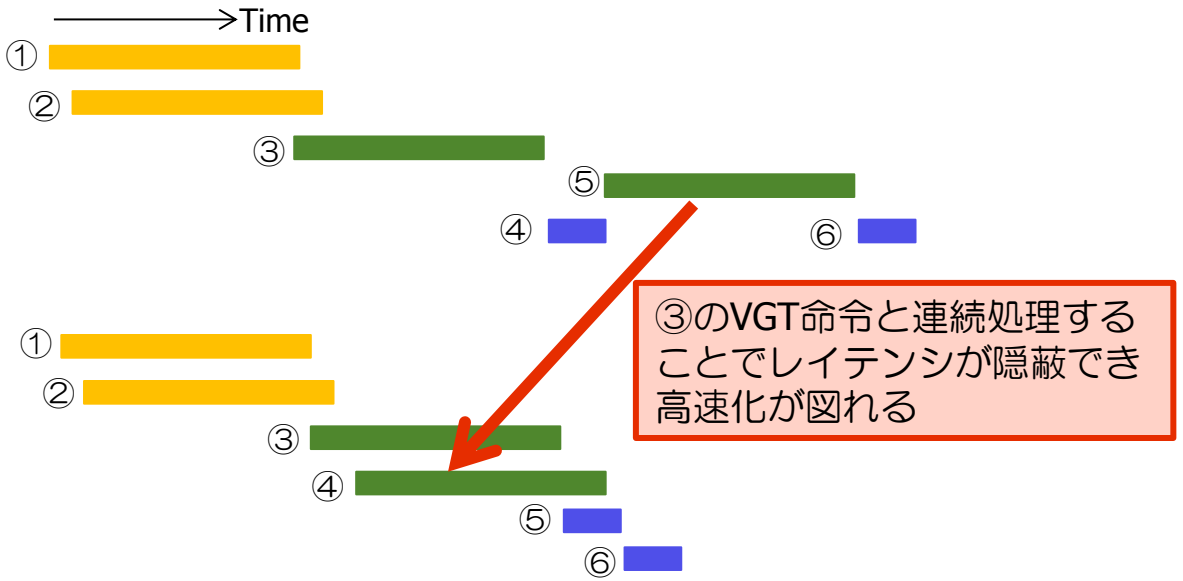
```
!CDIR GTHREORDER
DO I = 1, N
 A(IND(I)) = A(IND(I)) + X(I)
 A(IND(I)+1) = A(IND(I)+1) + X(I)
ENDDO
```

IND(I),IND(I)+1に同じデータがないことを指定

【GTHREORDER指示行なし】
命令列（サンプル）
① VLD　IND(I)
② VLD　X(I)
③ VGT　A(IND(I))
④ **VSC　A(IND(I))**
⑤ **VGT　A(IND(I)+1)**
⑥ VSC　A(IND(I)+1)

【GTHREORDER指示行あり】
命令列(サンプル)
① VLD　IND(I)
② VLD　X(I)
③ VGT　A(IND(I))
④ **VGT　A(IND(I)+1)**
⑤ **VSC　A(IND(I))**
⑥ VSC　A(IND(I)+1)

タイミングチャート(イメージ)



③のVGT命令と連続処理することでレイテンシが隠蔽でき高速化が図れる

NOCONFLICT[(配列変数指定,…)]指示行

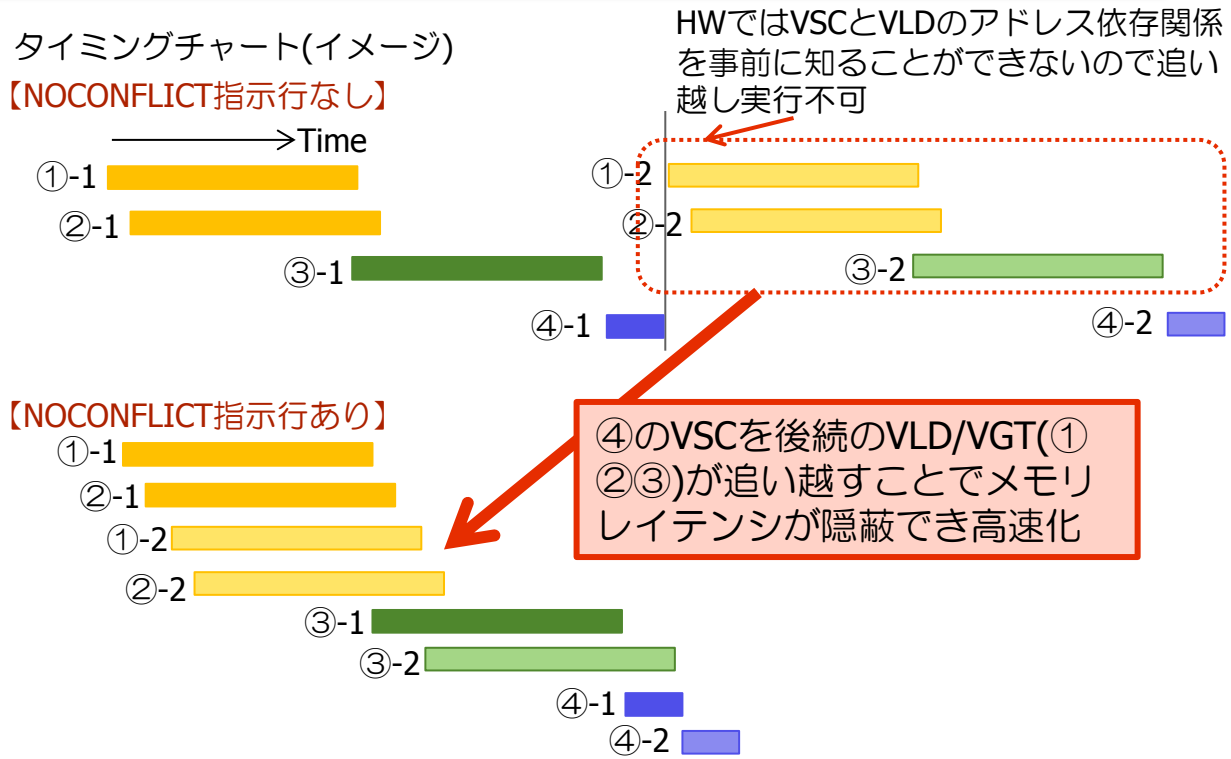
指定された配列(省略時はすべての配列)の定義と他の変数・配列参照にメモリ上の重なりがないことを指定し，実行時に後続の参照が配列定義の追い越しを行う(HWの命令追い越し)ことを許可する.

```
!CDIR NOCONFLICT
DO I = 1, 512
 A(IND(I)) = Y(IND(I)) + X(I)
ENDDO
```

配列:Aと配列:X,Y,INDにアドレスの重なりがないことを指定

命令列（サンプル）
I=1～256
①-1 VLD IND(I)
②-1 VLD X(I)
③-1 VGT Y(IND(I))
④-1 VSC **A(IND(I))**

I=257～512
①-2 VLD **IND(I)**
②-2 VLD **X(I)**
③-2 VGT **Y(IND(I))**
④-2 VSC A(IND(I))



VOVERTAKE[(配列変数指定,…)], VOB指示行(1/2)

- 直後の配列式またはDO ループ中のベクトルストア(連続，等間隔，リスト)を，後続のスカラロード，スカラストア，ベクトルロードが実行時に追い越して実行することを許可することを指定する.
- 配列変数指定がある場合，指定された配列のベクトルストアが追越しの対象となる. 指定がない場合は全て.
- NOCONFLICT指示行との違いは，コンパイラが重なりがありうると判断した場合でも許可すること. したがって危険性が大きい.
- VOB指示行は，VOVERTAKE指示行の効果を配列式・ループ終了後は許可しないと言う指定. VOB指示行を忘れると，ループを超えた追い越しが行われ結果不正となる場合があるので注意が必要(あえてループを超えた追い越しを使用する場合は，後ろのループでVOB指示行を指定する).
- ベクトル化を行うためには，NODEP指示行が必要.

VOVERTAKE[(**配列変数指定**,…)], VOB**指示行**(2/2)

```
!CDIR NODEP
!CDIR VOVERTAKE(A),VOB
DO I = 1, 512
 A(IND(I)) = A(IND(I)) + X(I)
ENDDO
```

配列:Aと配列**:X,IND,A**にアドレスの重なりがないことを指定

命令列（サンプル）

I=1～256
- ①-1 VLD　IND(I)
- ②-1 VLD　X(I)
- ③-1 VGT　A(IND(I))
- ④-1 VSC　**A(IND(I))**

I=257～512
- ①-2 VLD　IND(I)
- ②-2 VLD　X(I)
- ③-2 VGT　**A(IND(I))**
- ④-2 VSC　A(IND(I))

③と④でアドレスの依存関係がある場合はベクトル化不可
NODEP指示行でベクトル化

アドレスの依存関係なし
IND(I)の全要素で重なりがないことを保証する必要あり

補足(NODEP/NOCONFLICT/VOVERTAKEの違い)

コンパイラ指示行NODEP/NOCONFLICT/VOVERTAKEの相違点は以下の通り

指示行名	内　容	相違点
NODEP	左辺の配列と右辺の配列の間に依存関係がないことを指定する．コンパイラは依存関係がないとしてベクトル化を行う．	ベクトル化に関係するものであり、命令の追い越しには関係しない．
NOCONFLICT	配列式やDOループ中の指定された(省略するとすべてが対象)配列の定義と参照にメモリ上に重なりがないことを指定する．コンパイラは先行する定義より先に後続の参照を実行することを許可する．	ベクトル化に関係するものでもあり、かつ命令の追い越しに関係するものである．
VOVERTAKE	配列式やDOループ中のベクトルストアを，後続のスカラロード，スカラストア，ベクトルロードが追い越して実行することを許可する．	命令の追い越しに関係するものであり,ベクトル化には関係しない．

```
!CDIR NODEP(A)
DO I = 1, N
 A(I) = A(I+K) +X(I)
ENDDO
```

※右辺と左辺の配列Aの参照と定義にベクトル化を行っても結果の差異を生じる依存関係がないことを宣言する．

```
!CDIR NOCONFLICT
DO I = 1, 512
 A(IND(I)) = Y(IND(I)) + X(I)
ENDDO
```

※配列A,Y,Xの定義と参照にメモリ上の重なりがないことを指定し,後続の配列YやXの参照命令が先行する配列Aの定義を追い越すことを許可する．

```
!CDIR NODEP
!CDIR VOVERTAKE(A),VOB
DO I = 1, 512
 A(IND(I)) = A(IND(I)) + X(I)
ENDDO
```

※配列Aを引用するINDの全要素に重なりがない(配列Xとも重なりがない)ことを指定し,配列Aのロード命令がストア命令を追い越すことを許可する．

3. SX-ACEツールの紹介

SX-ACEで利用可能なツール

SX-ACE向けに最適化を実施する上で利用可能なツールは以下.

- 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・変形リスト)
- プログラム情報
- プロファイラ
- 簡易性能解析(ftrace,NEC Ftrace Viewer)

ベクトル化(並列化)診断メッセージ

コンパイルオプションに -Wf,-pvctl fullmsgを追加することにより，詳細なベクトル化診断メッセージおよび並列化診断メッセージを出力する．

```
 9: V------>          do i=1,n
10: |       A           a(i) = a(i) * b(i) - c(i)
11: |        S          if(a(i).le.0.d0) write(6,*) a(i)
12: V------           enddo
13:              c
14: +------>         do i=2,n
15: |                  a(i) = a(i-1) * b(i) - a(i-1) * c(i)
16: +------           enddo
```

```
 8 vec (  4): 配列式全体をベクトル化する。
 9 vec (  2): ループの一部をベクトル化する。
 9 vec ( 25): 領域の大きさが8000である作業ベクトルを使ってベクトル化を行う。
 9 vec ( 29): 配列bに対して ADB を使用する。
 9 vec ( 29): 配列cに対して ADB を使用する。
 9 vec ( 29): 配列aに対して ADB を使用する。
11 opt (1017): サブルーチン呼出しがあるため最適化できない。
11 vec ( 17): ベクトル化の対象とならない文である。
14 vec (  3): ベクトル化できないループである。
14 vec ( 13): ループ分割によるオーバヘッドが大きすぎる。
15 opt (1037): 異なる繰り返しで同一の配列要素を定義／参照している。
15 vec ( 20): aにベクトル化不可の依存関係がある。
```

- ベクトル化診断メッセージを採取することで，ベクトル化や自動並列化がされない要因を明らかにすることが可能．
- コンパイラの最適化情報(どの配列をADBに乗せたなど)を得る．

編集リストと変形リスト

編集リスト

- ベクトル化／自動並列化に関する情報をソースプログラムの左側に表示（コンパイラオプション　-R5，-R2）
- どのループがベクトル化されたか？
- どの文が部分的にスカラで実行されるか？

変形リスト

- 拡張ベクトル化で行ったループの変形結果のソースを元のソースプログラムとマージして表示（コンパイラオプション　-R1，-R2）
- どのようにループが変形されたか？

編集リスト (1/4)

（編集リストの例）

```
FILE NAME:t5.f
PROGRAM NAME: sub
FORMAT LIST

 LINE    LOOP     FORTRAN STATEMENT
    1:                subroutine sub(a,b,c,d,z,ix)
    2:                real,dimension(100)::a,b,c,d,x,y,z
    3:                integer ix(100)
    4: V------>         do I = 1,100
    5: |       I          call sub2(x,a,b,I)
    6: |                  y(I) = c(I) + d(I)
    7: |       S          z(ix(I)) = z(ix(I)) + x(I) + y(I)
    8: V------         enddo
    9:                end
```

編集リスト (2/4)

◆ループ全体がベクトル化される場合

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

◆ベクトル化されない場合

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

◆ループの一部がベクトル化される場合

```
V------>    do  I  =1,100
|             a(I)=b(I)+c(I)
|       S     print *,a(I)
V------     enddo
```

ベクトル化不可の処理がある行には
“S” が表示される

◆配列に対してADBが使用される場合

```
V------>    do  I=2,100
|       A     a(I)=b(I)+b(I-1)
V------     enddo
```

編集リスト (3/4)

配列式をベクトル化した場合 (1)

```
                real  a(90), b(90), c
 V=======       a(1:90)=b(1:90)+c
```

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

配列式をベクトル化した場合 (2)

```
                real  a(90), b(90), c
                integer  d(90), e(90)
V------>        a(1:90)=b(1:90)+c
|               d(1:90)=int(a(1:90))
V------         e(1:90)=d(1:90)+1
```

ループ融合している場合は、その範囲について“V”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

編集リスト (4/4)

◆多重ループが一重化された場合

```
W------>    do J =1,100
|*---->      do I =1,100
||             a(I,J)=b(I,J)+c(I,J)
|*----       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

◆ループの入れ換えが行われた場合

```
X------>    do J =1,1000
|+----->      do I =1,10
||             a(I,J)=b(I,J)+c(I,J)
|+-----       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

変形リスト

```
LINE                                FORTRAN STATEMENT
   1        subroutine sub(a,b)
   2        real,dimension(100,100)::a,b
   3        do j = 1,100
   4          do i = 1,99
   5            a(i+1,j)=a(i,j)+b(i,j)
   6          enddo
   7        enddo
   .          do i = 1,99
   .  !CDIR NODEP
   .            do j = 1,100
   .              a(i+1,j)=a(i,j)+b(i,j)
   .            enddo
   .          enddo
   9        end
```

ループが入れ換えられて

ベクトル化不可の依存関係がなくなった。

プログラム情報（1/2）

プログラム全体の性能情報を採取

- 環境変数F_PROGINFに YES または DETAILで表示
- 下記の例はDETAIL

```
    ******  Program Information  ******
(a) Real Time (sec)                :          0.429530
(b) User Time (sec)                :          0.428730
(c) Sys  Time (sec)                :          0.000722
(d) Vector Time (sec)              :          0.428555
(e) Inst. Count                    :        240708300.
(f) V. Inst. Count                 :        117679817.
(g) V. Element Count               :      30126031456.
(h) V. Load Element Count          :      10741743662.
(i) FLOP Count                     :      17179869321.
(j) MOPS                           :      70555.034495
(k) MFLOPS                         :      40071.535281
(l) A. V. Length                   :        255.999986
(m) V. Op. Ratio (%)               :         99.593282
(n) Memory Size (MB)               :        256.031250
(p) MIPS                           :        561.444965
(q) I-Cache (sec)                  :          0.000040
(o) O-Cache (sec)                  :          0.000061
Bank Conflict Time
(r)   CPU Port Conf. (sec)         :          0.000016
(s)   Memory Network Conf. (sec)   :          0.109435
(t) ADB Hit Element Ratio (%)      :         19.954172
```

平均ベクトル長、ベクトル演算率、バンクコンフリクト時間に着目

ループ長は十分か？

ベクトル演算率は十分か？

メモリアクセス性能の改善が必要か？

ADBは効果的に使用されているか？

プログラム情報(2/2)

a. 経過時間
b. ユーザ時間
c. システム時間
d. ベクトル命令実行時間
e. 全命令実行数
f. ベクトル命令実行数
g. ベクトル命令実行要素数
h. ベクトルロード要素数
i. 浮動小数点データ実行要素数
j. MOPS値
k. MFLOPS値
l. 平均ベクトル長
m. ベクトル演算率
n. メモリ使用量
o. MIPS値
p. 命令キャッシュミス時間
q. オペランドキャッシュミス時間
r. CPUポート競合時間
s. メモリネットワーク競合時間
t. ADBヒット率

プログラム情報（自動並列化時）（1/3）

```
 ******  Program Information  ******
Real Time (sec)                :          0.307168
User Time (sec)                :          1.190239
Sys  Time (sec)                :          0.007852
Vector Time (sec)              :          1.167033
Inst. Count                    :        241880273
V. Inst. Count                 :        117679849
V. Element Count               :      30126037402
V. Load Element Count          :      10741746602
FLOP Count                     :      17179869334
MOPS                           :      25415.263511
MFLOPS                         :      14433.966064
MOPS   (concurrent)            :     100019.632876
MFLOPS (concurrent)            :      56803.659976
A. V. Length                   :        255.999967
V. Op. Ratio (%)               :         99.589423
Memory Size (MB)               :        512.000000
Max Concurrent Proc.           :                 4
  Conc. Time(>= 1) (sec)       :          0.302443
  Conc. Time(>= 2) (sec)       :          0.301598
  Conc. Time(>= 3) (sec)       :          0.301181
  Conc. Time(>= 4) (sec)       :          0.285894
Event Busy Count               :                 0
Event Wait (sec)               :          0.000000
Lock Busy Count                :                 0
Lock Wait (sec)                :          0.000000
Barrier Busy Count             :                 0
Barrier Wait (sec)             :          0.000000
MIPS                           :        203.219919
MIPS (concurrent)              :        799.754906
I-Cache (sec)                  :          0.000145
O-Cache (sec)                  :          0.002183
Bank Conflict Time
  CPU Port Conf. (sec)         :          0.000239
  Memory Network Conf. (sec)   :          0.714319
ADB Hit Element Ratio (%)      :         20.310514
```

経過時間（秒）
ユーザ時間（秒）
システム時間（秒）
ベクトル命令実行時間（秒）
全命令実行数
ベクトル命令実行数
ベクトル命令実行要素数
ベクトルロード要素数
浮動小数点データ実行要素数
MOPS 値
MFLOPS 値
MOPS 値　（実行時間換算）
MFLOPS 値（実行時間換算）
平均ベクトル長
ベクトル演算率（%）
メモリ使用量（MB）
最大同時実行可能プロセッサ数
1台以上で実行した時間（秒）
2台以上で実行した時間（秒）
3台以上で実行した時間（秒）
4台以上で実行した時間（秒）
イベントビジー回数
イベント待ち時間（秒）
ロックビジー回数
ロック待ち時間（秒）
バリアビジー回数
バリア待ち時間（秒）
MIPS 値
MIPS 値（実行時間換算）
命令キャッシュミス（秒）
オペランドキャッシュミス（秒）
CPUポート競合時間（秒）
メモリネットワーク競合時間（秒）
ADBヒット率（%）

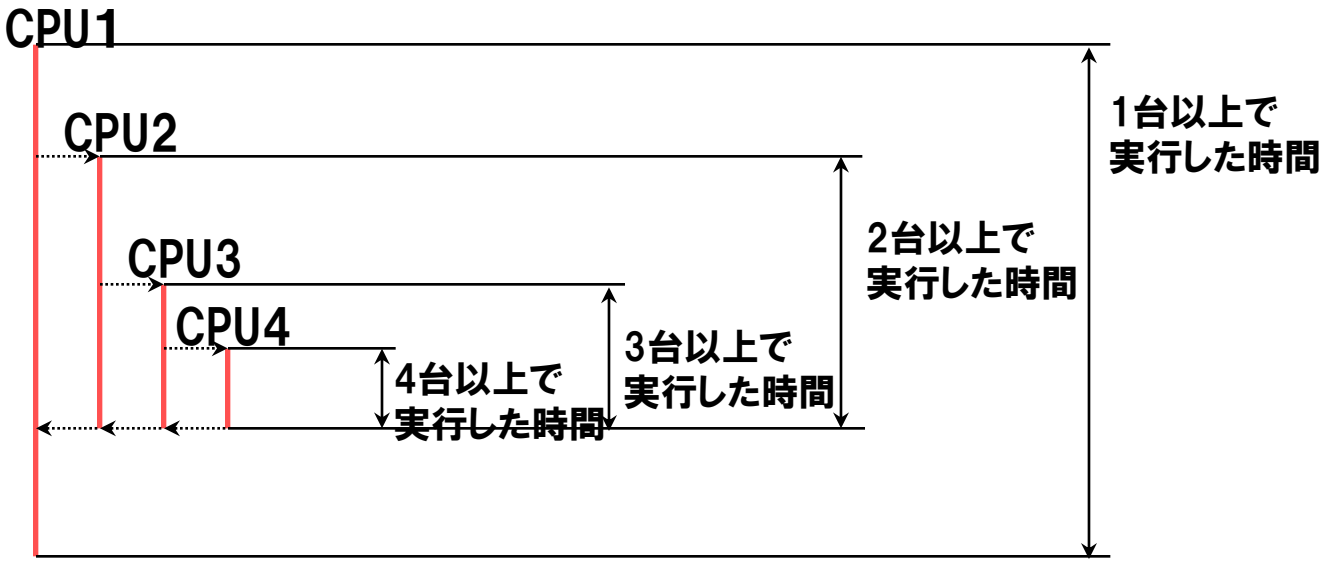
赤：並列処理使用時に表示

プログラム情報（自動並列化時）（2／3）

Conc. Time （Concurrent Time）

CPU n台以上で実行した時間

- プログラムが動作したCPUの時間を知ることができる
- 少なくとも1台のCPUが動いた時間、少なくとも2台のCPUが動いた時間、…を表す



プログラム情報（自動並列化時）（3／3）

Conc. Time（>=1）と比べ、 Conc. Time（>=2）が小さい

```
Conc. Time (>= 1) (sec):        74.154168
Conc. Time (>= 2) (sec):         8.549322
Conc. Time (>= 3) (sec):         8.292376
Conc. Time (>= 4) (sec):         8.071275
```

➔並列化率が低い　➔並列化されていないループを並列化

Conc. Timeの値に偏りがある

```
Conc. Time (>= 1) (sec):        69.503482
Conc. Time (>= 2) (sec):        58.271920
Conc. Time (>= 3) (sec):        33.497481
Conc. Time (>= 4) (sec):        12.927761
```

➔負荷バランスが悪い　➔ループ並列実行方法の変更

プログラム情報(MPI実行時)(1/6)

- MPIプログラム実行時のプログラム情報は環境変数MPIPROGINFを設定して採取
- プログラム情報をMPIプロセス毎に詳細に表示させたり，全MPIプロセスの情報を集計編集して表示させることが可能
- 表示は，MPIプログラムの実行において，MPI_FINALIZE手続きを呼び出した際にMPI_COMM_WORLD (MPIUNIVERSE=0) のランク0のMPIプロセスから標準エラー出力に対して行われる
- MPIPROGINFの値と表示内容は以下の通り
  - NO　　実行性能情報を出力しない(既定値)
  - YES　　基本情報を集約形式で出力
  - DETAIL　　詳細情報を集約形式で出力
  - ALL　　基本情報を拡張形式で出力
  - ALL_DETAIL　　詳細情報を拡張形式で出力

プログラム情報(MPI実行時)(2/6)

	Global Data of 4 processes	:	Min [U,R]	Max [U,R]	Average
a.	Real Time (sec)	:	0.023 [0,3]	0.035 [0,0]	0.029
b.	User Time (sec)	:	0.005 [0,3]	0.006 [0,1]	0.006
c.	Sys Time (sec)	:	0.003 [0,2]	0.003 [0,0]	0.003
d.	Vector Time (sec)	:	0.003 [0,3]	0.005 [0,1]	0.004
e.	Inst. Count	:	1262086 [0,3]	1393909 [0,1]	1338062
f.	V. Inst. Count	:	133790 [0,3]	142414 [0,1]	139146
g.	V. Element Count	:	33639232 [0,3]	33677845 [0,0]	33654603
h.	V. Load Element Count	:	20154 [0,3]	31174 [0,1]	25584
i.	FLOP Count	:	400 [0,1]	431 [0,0]	408
j.	MOPS	:	5404.267 [0,1]	7658.046 [0,3]	6243.678
k.	MFLOPS	:	0.062 [0,1]	0.088 [0,3]	0.073
l.	A. V. Length	:	236.316 [0,1]	251.433 [0,3]	242.001
m.	V. Op. Ratio (%)	:	96.415 [0,1]	96.755 [0,3]	96.560
	Total Memory Size (MB)	:	256.031 [0,0]	256.031 [0,0]	256.031
n.	Memory Size (MB)	:	192.031 [0,0]	192.031 [0,0]	192.031
o.	Global Memory Size (MB)	:	64.000 [0,0]	64.000 [0,0]	64.000
p.	MIPS	:	215.809 [0,1]	277.993 [0,3]	238.589
q.	I-Cache (sec)	:	0.000 [0,3]	0.000 [0,0]	0.000
r.	O-Cache (sec)	:	0.000 [0,3]	0.000 [0,1]	0.000
	Bank Conflict Time				
s.	CPU Port Conf. (sec)	:	0.000 [0,3]	0.000 [0,1]	0.000
t.	Memory Network Conf. (sec)	:	0.000 [0,3]	0.000 [0,1]	0.000
u.	ADB Hit Element Ratio (%)	:	0.000 [0,0]	0.000 [0,0]	0.000

プログラム情報(MPI実行時)(3/6)

a. 経過時間
b. ユーザ時間
c. システム時間
d. ベクトル命令実行時間
e. 全命令実行数
f. ベクトル命令実行数
g. ベクトル命令実行要素数
h. ベクトルロード要素数
i. 浮動小数点データ実行要素数
j. MOPS値
k. MFLOPS値
l. 平均ベクトル長
m. ベクトル演算率
n. メモリ使用量
o. グローバルメモリ使用量
p. MIPS値
q. 命令キャッシュミス時間
r. オペランドキャッシュミス時間
s. CPUポート競合時間
t. メモリネットワーク競合時間
u. ADBヒット率

プログラム情報(MPI実行時)(4/6)

- 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するには環境変数MPICOMMINFを設定する
- MPI_COMM_WORLD (MPI_UNIVERSE=0) のランク0のMPIプロセスがMPI_FINALIZE手続き中で標準エラー出力に対して行う
- MPICOMMINFの値と表示内容は以下の通り
  - NO　　通信情報を出力しない(既定値)
  - YES　　最小値，最大値，および平均値を表示
  - ALL　　最小値，最大値，平均値，および各プロセス毎の値を表示

プログラム情報(MPI実行時)(5/6)

出力例
(YES指定時)

MPI Communication Information:

Real MPI Idle Time (sec)	:	0.008 [0,0]	0.192 [0,3]	0.140
User MPI Idle Time (sec)	:	0.006 [0,0]	0.192 [0,3]	0.140
Total real MPI Time (sec)	:	0.305 [0,3]	0.366 [0,0]	0.329
Send count	:	0 [0,0]	11 [0,1]	8
Recv count	:	0 [0,1]	33 [0,0]	8
Barrier count	:	0 [0,0]	0 [0,0]	0
Bcast count	:	0 [0,0]	0 [0,0]	0
Reduce count	:	0 [0,0]	0 [0,0]	0
Allreduce count	:	0 [0,0]	0 [0,0]	0
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
Redscatblk count	:	0 [0,0]	0 [0,0]	0
Gather count	:	0 [0,0]	0 [0,0]	0
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
Number of bytes sent	:	0 [0,0]	4400000000 [0,1]	3300000000
Number of bytes recv	:	0 [0,1]	13200000000 [0,0]	3300000000
Put count	:	0 [0,0]	0 [0,0]	0
Get count	:	0 [0,0]	0 [0,0]	0
Accumulate count	:	0 [0,0]	0 [0,0]	0
Number of bytes put	:	0 [0,0]	0 [0,0]	0
Number of bytes got	:	0 [0,0]	0 [0,0]	0
Number of bytes accum	:	0 [0,0]	0 [0,0]	0

プログラム情報（MPI実行時）（5/6）

- MPICOMMINF利用時の注意事項
- 本機能は，プロファイル版MPIライブラリをリンクした場合に利用可能
- プロファイル版MPIライブラリは，MPIプログラムのコンパイル/リンク用コマンド(mpisxf90等)の `-mpitrace`，`-mpiprof`，`-ftrace`のいずれかのオプション指定によりリンクされる

プログラム情報(MPI実行時)(6/6)

標準出力および標準エラー出力の出力先をmpisep.shと環境変数MPISEPSELECTを用いて制御する

- 値が1の時，標準出力だけstdout.$IDに出力される
- 値が2の時，標準エラー出力だけがstderr.$IDに出力される(既定値)
- 値が3の時，標準出力はstdout.$IDに，標準エラー出力はstderr.$IDに出力される
- 値が4の時，標準出力および標準エラー出力が，std.$IDに出力される
- その他の時，標準出力も標準エラー出力もファイルに出力しない

/usr/lib/mpi/mpisep.sh

```
#!/sbin/sh
ID=$MPIUNIVERSE:$MPIRANK
case ${MPISEPSELECT:-2} in
1) exec $* 1>> stdout.$ID                  ;;
2) exec $*                 2>> stderr.$ID ;;
3) exec $* 1>> stdout.$ID 2>> stderr.$ID ;;
4) exec $* 1>> std.$ID    2>&1             ;;
*) exec $* ;;
esac
```

- mpisep.shの使用例(値=3を指定する場合)

```
#PBS -v MPISEPSELECT=3
mpirun -np 4 /usr/lib/mpi/mpisep.sh a.out
```

プロファイラ情報

システム関数やライブラリの処理時間を知ることが可能

```
> sxf90 -p test.f          -p を指定してコンパイル＋リンク
> a.out
> prof a.out
   ①     ②        ③        ④          ⑤           ⑥
%Time  Seconds  Cumsecs   #Calls    msec/call     Name
 35.0    56.80    56.80    1320         43.0303    sub7_
 28.2    45.73   102.53      20       2286.5       func15_
 12.8    20.80   123.34                            ev_cdexp ← システムの手続
  6.6    10.64   133.98      40        266.0       func23_
  5.5     8.86   142.84   15000          0.0005    sub12_
    :       :       :       :             :          :
```

①：全体に占める手続毎のCPU時間の割合（％）
②：手続毎のCPU時間（秒）
③：先頭からの合計のCPU時間（秒）
④：手続の呼出し回数（-pを指定しない手続やシステムルーチンに対しては表示されない）
⑤：一回の呼出し毎のCPU時間（〃）
⑥：手続の入口名（ユーザ手続の場合、最後に_が付加される）

プロファイラ情報(自動並列化時)

自動並列化時の出力情報

- 並列実行されたサブルーチンの各タスクでの実行時間
  - 自動並列化されたループの実行時間
    - ➔ 元のサブルーチン名＋＄１、＄２…のCPU時間
  - スピンウェイト時間
    - ➔??_?pmpark, ??_?pmret のCPU時間

```
Root1 :
 %Res.      T/M    Micro(busy-wait)      CPU  #Calls    msec/call  Name
   0.0     0.00                        41.16                      ex_ipmpark
   0.0             41.16                                           -micro1
   0.0     0.00                        34.93    5428       6.4352  slave$1_
  13.6             22.29                      2714       8.2130   -micro1
   7.7             12.64                      2714       4.6573   -micro2
   0.0     0.00                        32.48                      ex_ipmret
   0.9              1.49                                           -micro1
  19.0             30.99                                           -micro2
   0.0     0.00                        17.32     282      61.4184  shl24s$1_
   5.3              8.71                       141      61.7730   -micro1
   5.3              8.61                       141      61.0641   -micro2
```

簡易性能解析(ftrace)（1/3）

利用の方法

```
% sxf90 -ftrace test.f90
ジョブの投入
% sxftrace
```

or

```
% sxf90 -ftrace test.f90
環境変数setenv F_FTRACE YES
ジョブ投入
```

実行後,カレントディレクトリにftrace.out(解析情報ファイル)が作成される

注意事項

-ftrace指定でコンパイルされた手続から-ftrace指定なしでコンパイルされた手続を呼び出す場合,呼び出し回数以外の測定値は呼び出し先の手続の性能情報を含んだ値となる

例：

sub1.f

```
SUBROUTINE SUB1
CALL SUB2(X,Y,Z)
END
```

sub2.f

```
SUBROUTINE SUB2(X,Y,Z)
X=SQRT(Y**2+Z**2)
END
```

sub1.f のみを-ftraceでコンパイルした場合、ftraceリストにはsub2の情報は表示されず,sub1の性能情報にsub2の情報を含んだ値が表示される.

簡易性能解析(ftrace)（2/3）

サブルーチン・ユーザ関数単位にプログラム情報を表示

呼出し回数

呼出したプログラムを含まない実行時間とその割合

1回あたりの EXCLUSIVE TIME

proginf と同じ

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

Execution Date : Thu Jan 15 15:52:50 2015
Total CPU Time : 0:04'38"876 (278.876 sec.)
```

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.%
sub5	15792	271.375 (97.3)	17.184	843.1	2.3	18.77	136.0	4.106	7.603	27.078	0.000	0.000	0.00
sub3	1616	2.779 (1.0)	1.719	842.7	2.3	18.77	136.0	0.042	0.078	0.278	0.000	0.000	0.00
sub9	1178190	2.184 (0.8)	0.002	11087.6	0.0	97.32	100.0	1.737	0.000	0.000	0.972	0.001	99.85
sub10	1178190	2.004 (0.7)	0.002	29645.8	11756.1	99.14	250.0	1.282	0.000	0.000	0.919	0.000	99.86
main_	1	0.380 (0.1)	380.484	281.8	0.0	0.00	0.0	0.000	0.001	0.004	0.000	0.000	0.00
sub2	1235	0.151 (0.1)	0.122	16496.3	0.0	99.06	250.0	0.151	0.000	0.000	0.059	0.062	4.33
sub1	1235	0.001 (0.0)	0.001	3950.2	0.0	96.43	250.0	0.000	0.000	0.000	0.000	0.000	0.00
sub7	1616	0.001 (0.0)	0.000	129.6	0.0	40.00	10.0	0.000	0.000	0.000	0.000	0.000	88.71
sub4	16	0.000 (0.0)	0.018	799.9	2.2	18.74	136.0	0.000	0.000	0.000	0.000	0.000	0.00
sub8	16	0.000 (0.0)	0.013	1954.5	0.0	79.84	10.0	0.000	0.000	0.000	0.000	0.000	0.64
sub6	16	0.000 (0.0)	0.000	10685.6	4211.1	98.52	250.0	0.000	0.000	0.000	0.000	0.000	4.80
total	2377923	278.876 (100.0)	0.117	1138.1	86.8	40.44	160.4	7.319	7.683	27.360	1.950	0.064	95.09

簡易性能解析(ftrace)（3/3）

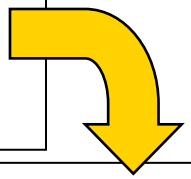
ユーザ指定リージョン

- 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)
```



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

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

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
- sxftrace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

```
% sxftrace -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）

```
% sxftrace -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

```
% sxftrace -f ftrace.out.* -all
```

簡易性能解析(ftrace)（MPI実行時）(2/3)

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

Execution Date : Fri Jan  9 16:20:54 2015 (a)
Total CPU Time : 0:00'09"011 (9.011 sec.) (b)
```

(c) FREQUENCY	(d) EXCLUSIVE TIME[sec](%)	(e) AVER.TIME [msec]	(f) MOPS	(g) MFLOPS	(h) V.OP RATIO	(i) AVER. V.LEN	(j) VECTOR TIME	(k) I-CACHE MISS	(l) O-CACHE MISS	(m) BANK CONFLICT CPU PORT	(n) BANK CONFLICT NETWORK	(o) ADB HIT ELEM.%	(p) PROC.NAME
1	9.011(100.0)	9010.675	5827.4	0.0	98.30	224.0	5.839	0.002	0.766	0.004	0.849	0.00	example3
1	9.011(100.0)	9010.675	5827.4	0.0	98.30	224.0	5.839	0.002	0.766	0.004	0.849	0.00	total
31	7.136(79.2)	230.189	1991.3	0.0	97.00	166.8	4.965	0.001	0.012	0.000	0.641	0.00	wait
93	0.000(0.0)	0.005	235.5	0.0	24.66	31.0	0.000	0.000	0.000	0.000	0.000	0.00	irecv

(q) ELAPSED TIME[sec]	(r) COMM.TIME [sec]	(s) COMM.TIME / ELAPSED	(t) IDLE TIME [sec]	(u) IDLE TIME / ELAPSED	(v) AVER.LEN [byte]	(w) COUNT	(x) TOTAL LEN [byte]	PROC.NAME
9.049	8.179	0.904	0.024	0.003	381.5M	93	34.6G	example3
7.136	7.136	1.000	0.024	0.003	381.5M	93	34.6G	wait
0.001	0.000	0.835	0.000	0.000	0.0	0	0.0	irecv

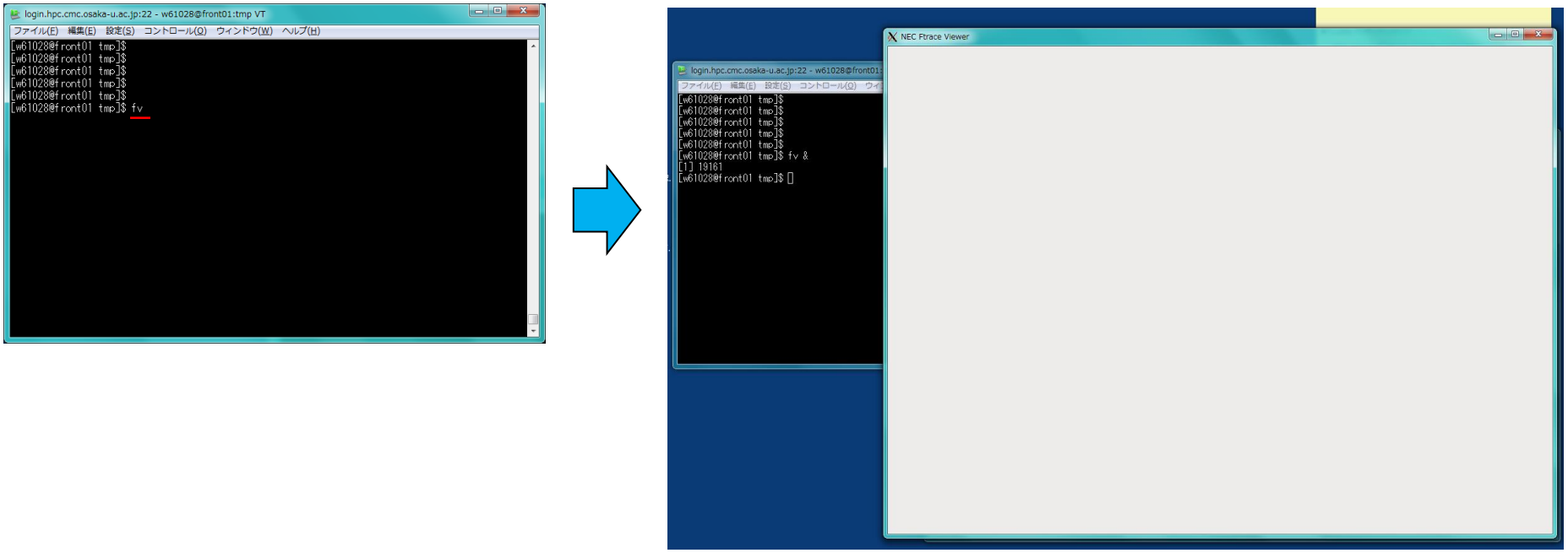
※この情報がMPI実行時のみ出力されるMPI通信の情報

簡易性能解析(ftrace)（MPI実行時）（3/3）

a. プログラムが終了した日時
b. 各プログラム単位での CPU 時間の合計
c. プログラム単位の呼び出し回数
d. プログラム単位の実行に要した EXCLUSIVE な CPU 時間(秒)と，それのプログラム全体の実行に要した CPU 時間に対する比率
e. プログラム単位の 1 回の実行に要した平均 CPU 時間(ミリ秒)
f. MOPS 値
g. MFLOPS 値
h. ベクトル演算率
i. 平均ベクトル長
j. ベクトル命令実行時間(秒)
k. 命令キャッシュミス時間(秒)
l. オペランドキャッシュミス時間(秒)
m. メモリアクセスにおけるCPUポート競合時間(秒)
n. メモリアクセスにおけるメモリネットワーク競合時間(秒)
o. プログラム単位名(二次入口名の場合は主入口名) なお，*OTHERS* は緒元の制限で個別に測定できなかった手続がある場合にその累計を表す．また，最後の行の total はプログラム全体を表す
p. ADBヒット率(%)
q. 経過時間(秒)
r. MPI 通信処理時間(MPI 手続きの実行に要した時間，通信待ち時間(r) を含む) (秒)
s. (p)と経過時間に対する比率
t. MPI 通信処理中における通信待ち時間(秒)
u. (r)と経過時間に対する比率
v. MPI 通信一回当たりの平均通信時間 (byte，Kbyte，Mbyte，Gbyte，Tbyte または Pbyte)
w. MPI 通信回数
x. MPI 通信の通信量 (byte，Kbyte，Mbyte，Gbyte，Tbyte または Pbyte)

NEC Ftrace Viewer（1/6）

- **簡易性能解析**(ftrace)**の結果をGUIを用いて，視覚的に行う．**
- X Window**サーバが必要(ネットワーク上問題があればセンターに要相談)**
- GUI **画面の表示**
  - **“fv”コマンドの実行**
  - **ウィンドウが立ち上がり，**NEC Ftrace Viewer**画面が表示される．**



NEC Ftrace Viewer（2/6）

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

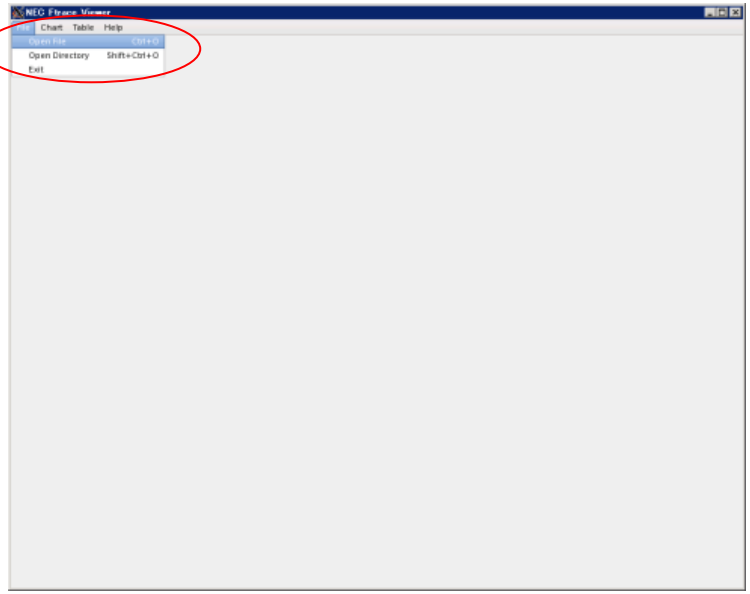
- Open File
  - 指定した ftrace.out もしくは ftrace.out.X.X を1つ読み込む.
- Open Directory
  - 指定したディレクトリ直下の ftrace.out もしくは ftrace.out.X.X を全て読み込む.

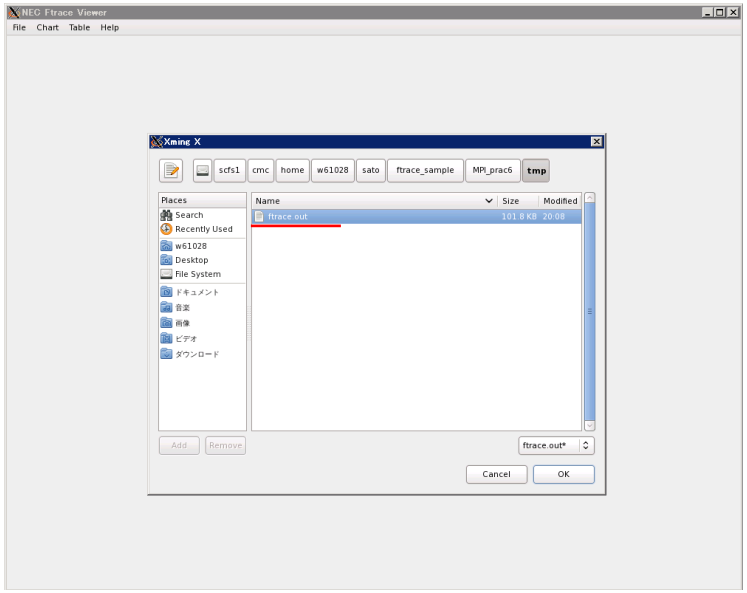
  ※ ftrace.out と ftrace.out.X.X が同じディレクトリにある場合，読み込みに失敗する.

NEC Ftrace Viewer（3/6）

逐次または自動並列実行の場合(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

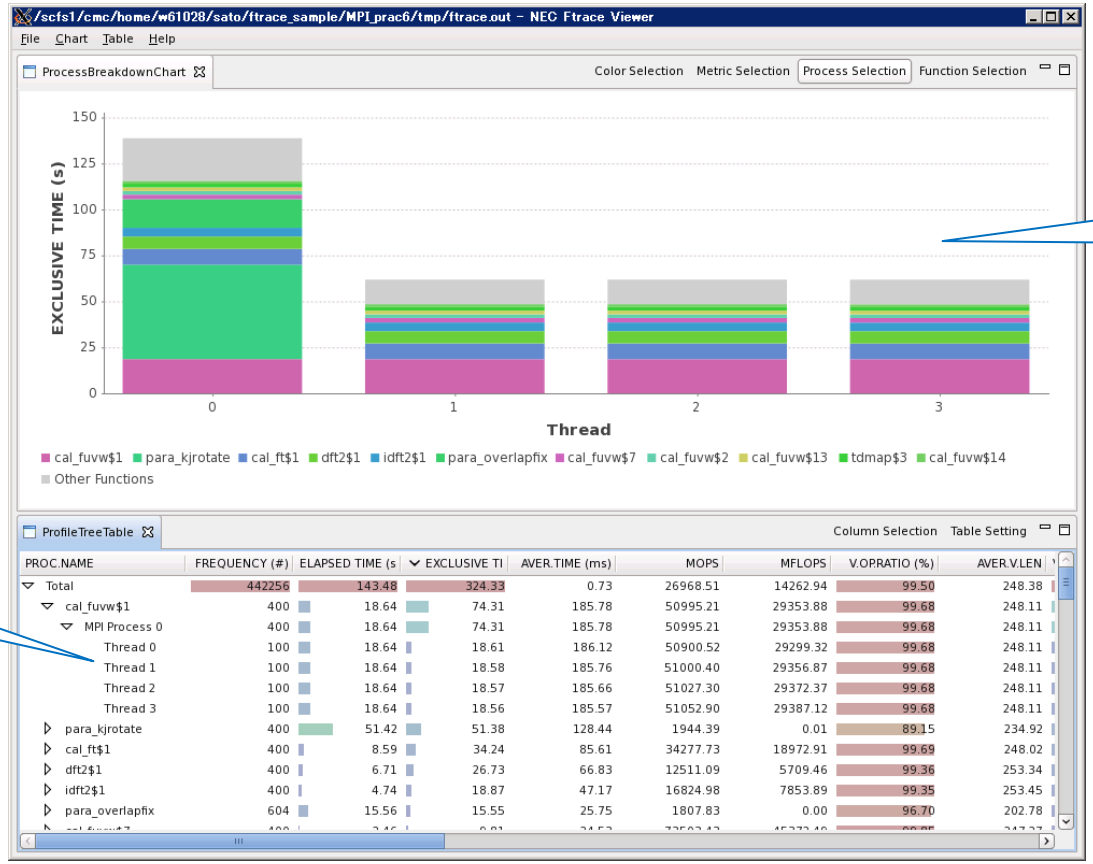




NEC Ftrace Viewer（4/6）

逐次または自動並列実行の場合(2/2)

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



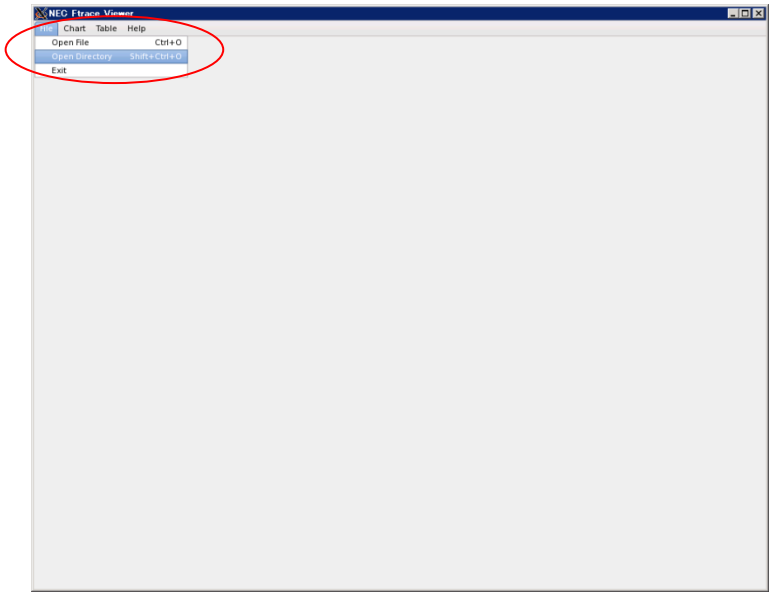
右クリックでグラフを
画像として保存できます

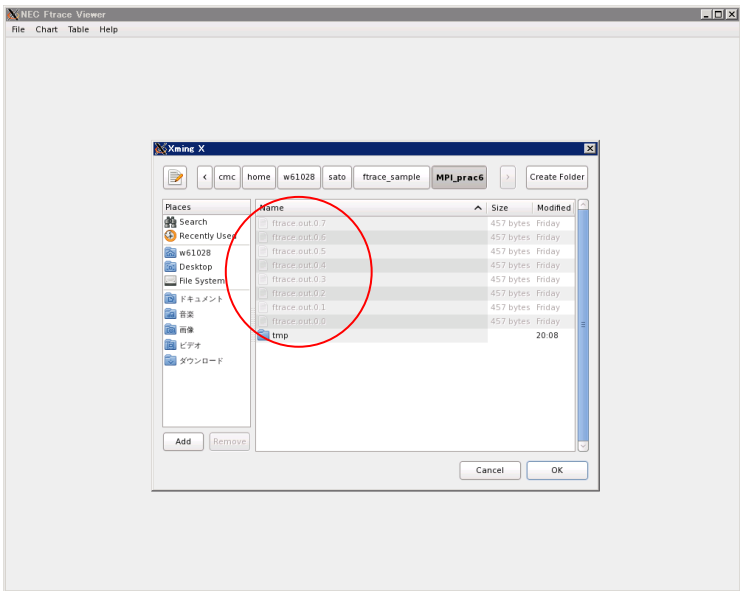
タスクごとの性能情報が
表示されます

NEC Ftrace Viewer（5/6）

MPI実行の場合(1/2)

- ftrace.out.X.X ファイルの読み込み
  - 「File」→「Open Directory」から読み込みたい ftrace.out.X.X があるフォルダを選択して「OK」を押下
    - ✓ 以下は，MPIプロセス分の ftrace.out ファイルを読み込む場合の例．
    - ✓ 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.X.X を指定する．

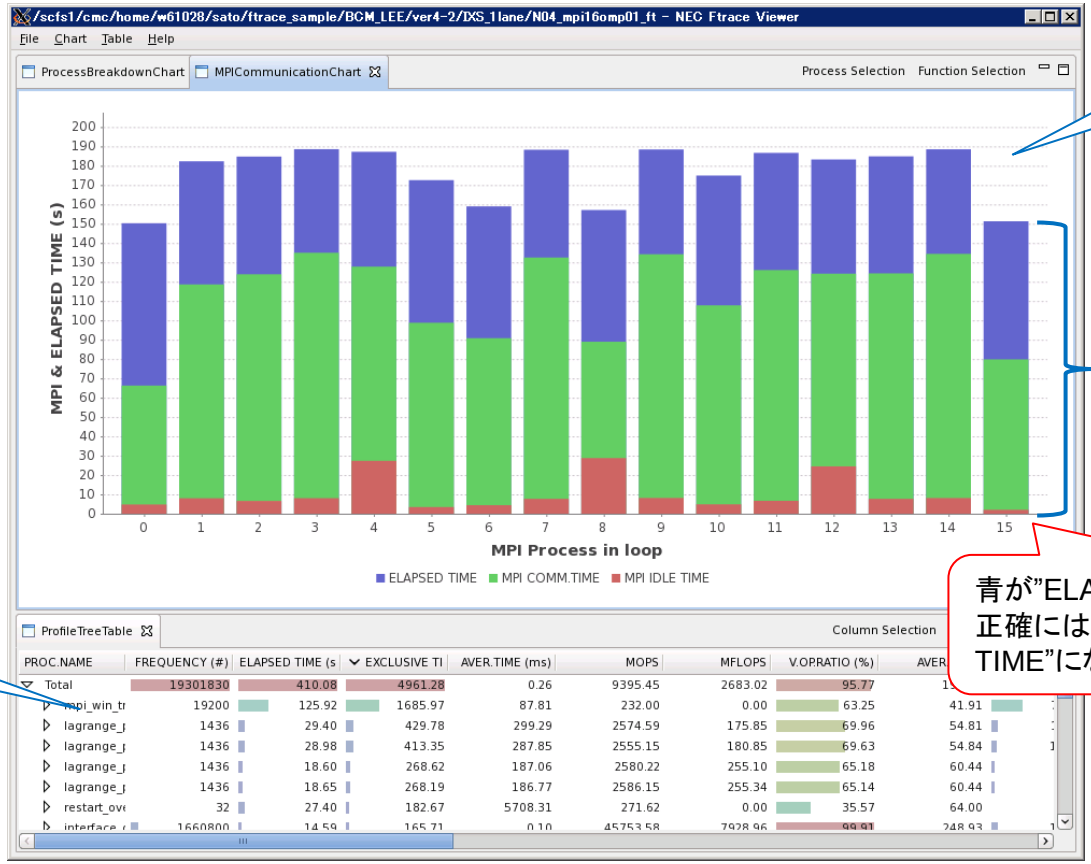




NEC Ftrace Viewer（6/6）

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る

ELAPSED TIME

MPI並列プロセスごとの
性能情報が表示される

青が”ELAPSED TIME”になっているが，
正確には青＋緑+赤が”ELAPSED
TIME”になる.

演習問題1

行列積の計算プログラムのMPIコードを用いて，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 TUNE/practice_1

% ./comp.sh

% qsub run.sh

- **実行が終了すると，**stderr.X:X(**標準エラー出力**)**ファイルと**stdout.X: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 stderr.0:0 (**プログラム情報の表示**)

% cat stdout.0:0 (**実行結果の表示**)

演習問題1(つづき)

② 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sxftrace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

% sxftrace –f ftrace.out.0.0

% sxftrace –f ftrace.out.*

③ X Windowを利用できる方は，Ftrace Viewerで表示してみてください．

% fv

演習問題1(つづき)

④ **エディタでソースコードを編集し，**FTRACE_REGION_BEGIN/_END**を挿入します．**

% vi(**エディタ**) sample1.f

```
26        t1=MPI_WTIME()
          call ftrace_region_begin( “loop” )
27        do j=ist,ied
28          do k=1,n
29            do i=1,n
30              a(i,j)=a(i,j)+b(i,k)*c(k,j)
31            end do
32          end do
33        end do
          call ftrace_region_end( “loop” )
34        call MPI_GATHER(a(1,ist),n*n2,MPI_REAL8,d,n*n2
35       &                ,MPI_REAL8,0,MPI_COMM_WORLD,  ierr)
```

⑤ **編集が終了したら，再度コンパイルしてジョブ投入してください．**

% ./comp.sh

% qsub run.sh

% sxftrace –f ftrace.out.*

4. SX-ACEチューニングのポイント

SX-ACEチューニングのポイント

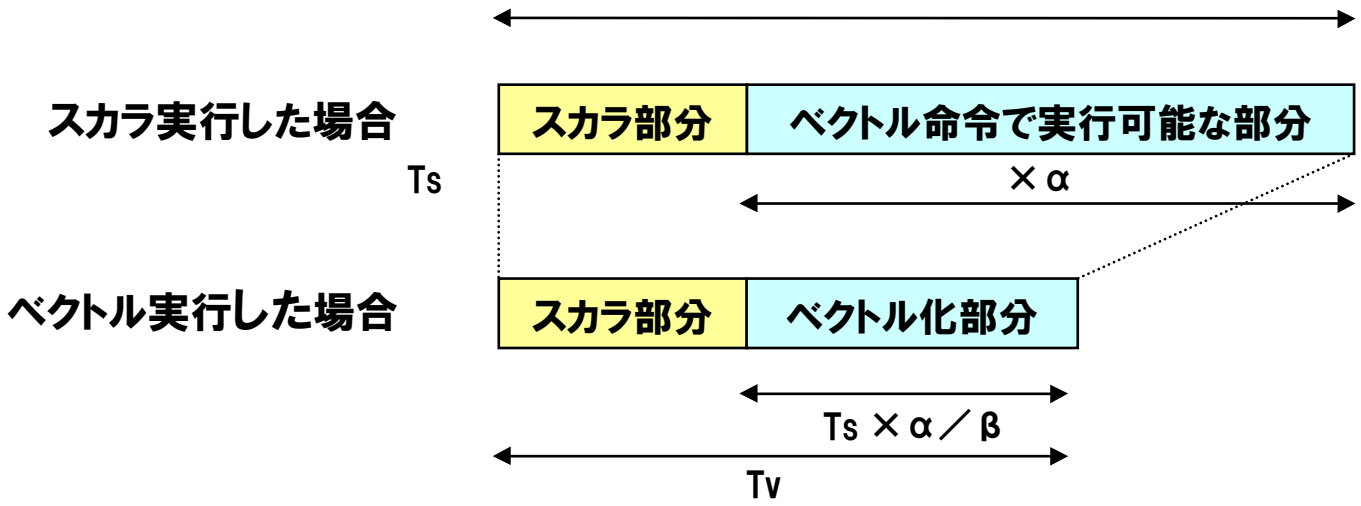
ベクトル型スーパーコンピュータであるSX-ACEのチューニングのポイントは以下の3点.

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）

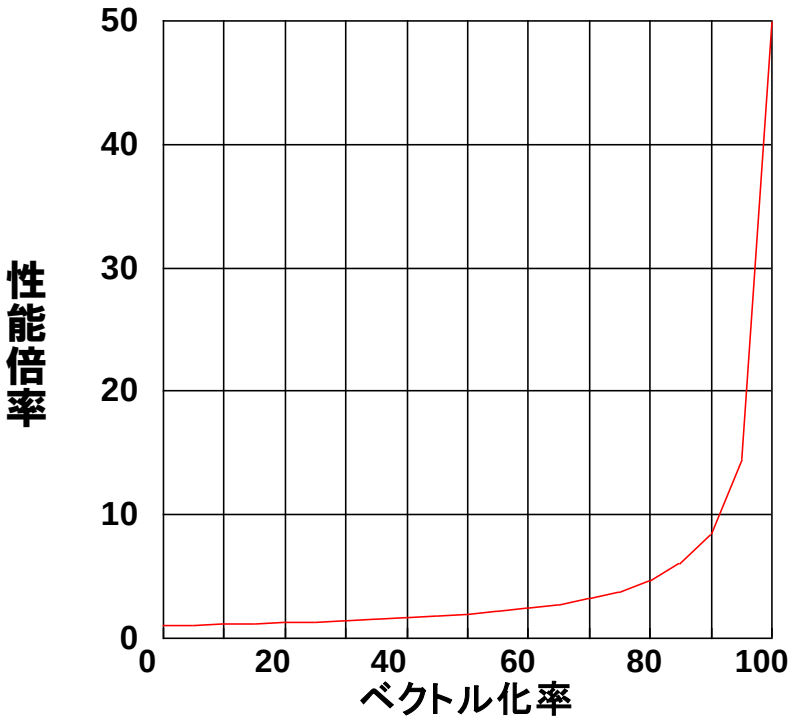
ベクトル演算率＝　(Ts＊α／β)／Tv

＝ α／（α＊(1－β)＋β）

一般にベクトル化率を正確に求めることは困難であるため，
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(アムダールの法則)

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

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「-R5 -Wf,-pvctl fullmsg」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: V------>         do i=1,n
 8: |       A        a(i) = a(i) * b(i) - c(i)
 9: |       S        if(a(i).le.0.d0) write(6,*) a(i)
10: V------          enddo

7  vec  (   2): Partially vectorized loop.
9  vec  (  17): Unvectorizable statement.
```

※ベクトル化を阻害する文（write）があるため部分ベクトル化

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

16  vec  (   3): Unvectorized loop.
17  opt  (1017): Subroutine call prevents optimization
```

※ベクトル化を阻害するサブルーチンcall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  (   3): Unvectorized loop.
13  vec  (  20): 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

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

23  vec  (   2): Partially vectorized loop.
24  vec  (  22): 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: V------>          do i=1,n
10: |       A           a(i) = a(i) * b(i) - c(i)
11: |        S          if(a(i).le.0.d0) write(6,*) a(i)
12: V------           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: |       A           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: V------>          do i=1,icnt
17: |        S          write(6,*) a(itbl(i))
18: V------           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

-Wf,-pvctl fullmsg を指定することにより、以下のメッセージが出力される

```
f90: vec(3): test.f, line 3: ベクトル化できないループである。
f90: vec(13): test.f, line 3: ループ分割によるオーバヘッドが大きすぎる。
f90: opt(1019): test.f, line 5: スカラ変数が異なる繰り返しで定義した値を参照 している。
f90: vec(21): test.f line 5: ベクトル化不可の依存関係がある。
f90: vec(21): test.f line 7: ベクトル化不可の依存関係がある。
```

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

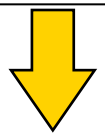
手続きのインライン展開

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
```

```
f90: vec(3): test.f, line 3: ベクトル化できないループである。
f90: opt(1025): test.f, line 4: 最適化を阻害する関数呼出しがある。
f90: vec(10): test.f, line 4: ループまたは配列式全体をベクトル化不可とする手続funが指定された
```



コンパイル時オプション -pi 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

-pi 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
```

```
f90: vec(1): test.f, line 3: ループ全体をベクトル化する。
f90: vec(24): test.f, line 3: ループの繰り返し数を最大5000と仮定する。
f90: opt(1222): test.f, line 4: 手続呼び出しをインライン展開した。
```

配列の重なりがない場合(NODEP指示行)

NODEP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．
-Wf,-pvctl listvec 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…

-Wf,-pvctl fullmsg を指定することにより、以下のメッセージが出力される

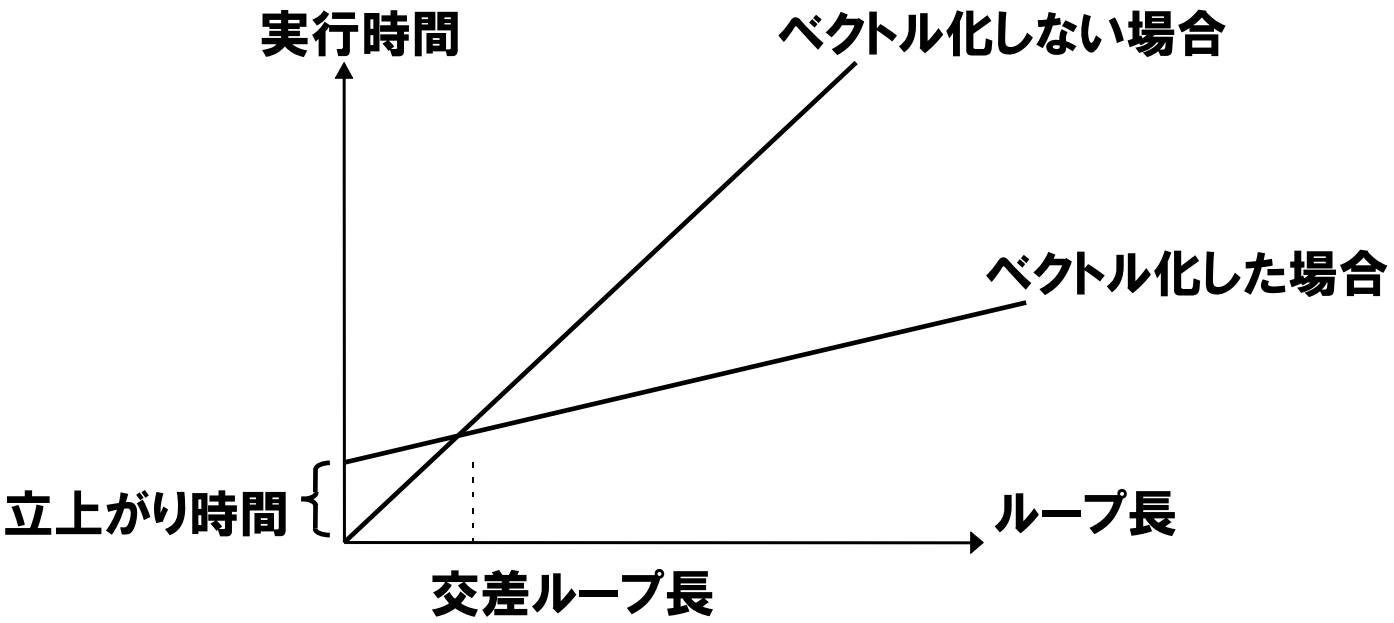
```
f90: vec(1): test.f, line 5: ループ全体をベクトル化する。
f90: vec(24): test.f, line 5: ループの繰り返し数を最大5000と仮定する。
f90: opt(1036): test.f, line 6: 異なる繰り返しで定義された値を参照している可能性
                                がある。(nosync/nodepを指定すれば最適化を行う)
f90: vec(26): test.f, line 6: List Vectorのマクロ演算としてベクトル化を行う。
```

```
!CDIR NODEP
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NODEP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nodep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,ベクトル化しないほうが速い.
(交差ループ長 5 程度)
ループ長をできるだけ長くした方が,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-ACE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-ACE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: |           !cdir noloopchg
29: |V----->       do j = 1,25600
30: ||                d1(i,j) = d1(i,j)+a1(i,j)+b1(i,j)*c1(i,j)
31: |V-----        enddo
32: +------       enddo
```

Loop2
- 短ベクトル
  →ループ長64
- 連続アクセス

```
36: X------>      do i = 1, 64
37: |+------>      do j = 1, 25600
38: ||               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，
SX-ACEだとバンク競合時間が増大

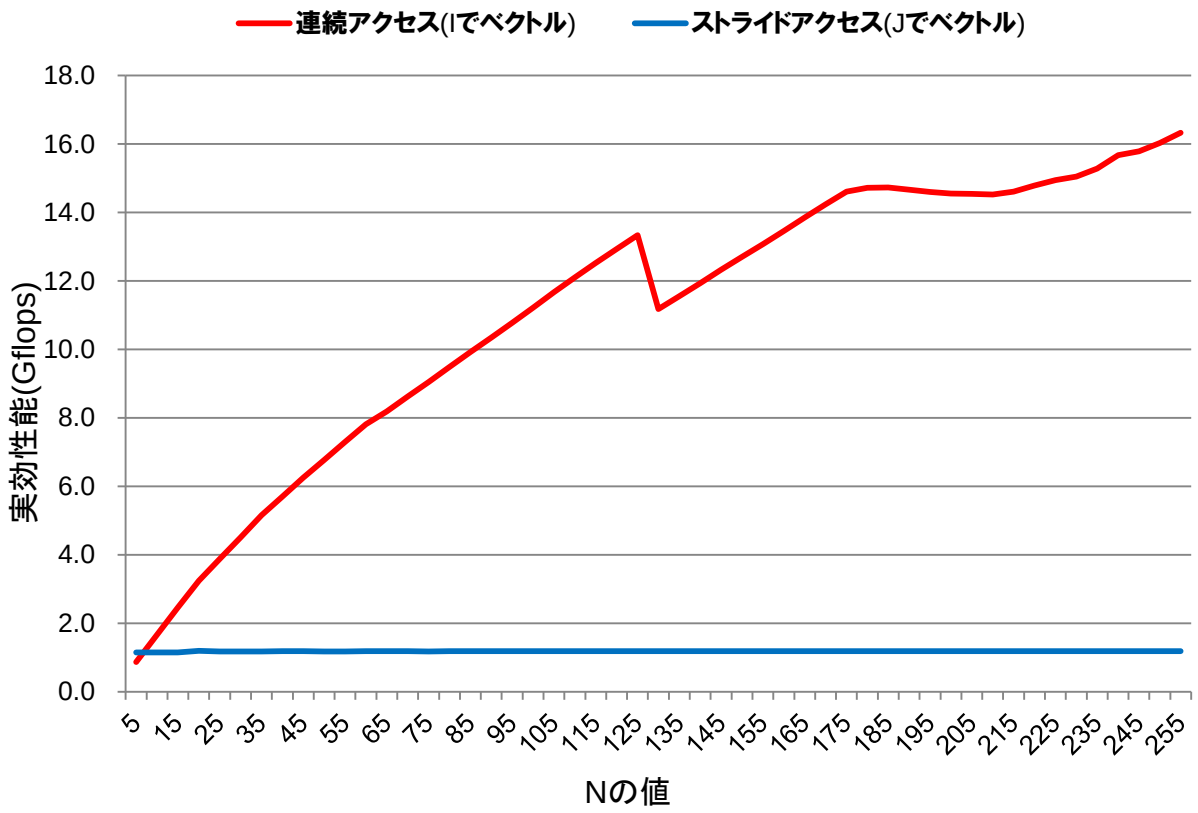
FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	ADB HIT ELEM.%	PROC.NAME
20000	9.836(22.6)	0.492	26747.4	9994.5	99.64	256.0	9.823	0.000	0.000	4.429	7.445		loop1(SX-9)
20000	83.800(73.0)	4.190	3138.6	1173.1	99.67	256.0	83.797	0.000	0.000	3.695	80.709	0.00	loop1(SX-ACE)
20000	10.611(24.3)	0.531	25140.1	9264.6	98.27	64.0	10.596	0.000	0.001	0.277	8.056		loop2(SX-9)
20000	9.172(8.0)	0.459	29083.9	10718.0	98.27	64.0	9.169	0.000	0.000	0.000	3.577	0.00	loop2(SX-ACE)

指示行の挿入によるループ長の拡大 (1/2)

SX-ACE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: |+------>       do i = 1, N
38: ||                d1(i,j) = d1(i,j)+a1(i,j)+b1(i,j)*c1(i,j)
39: |+------        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



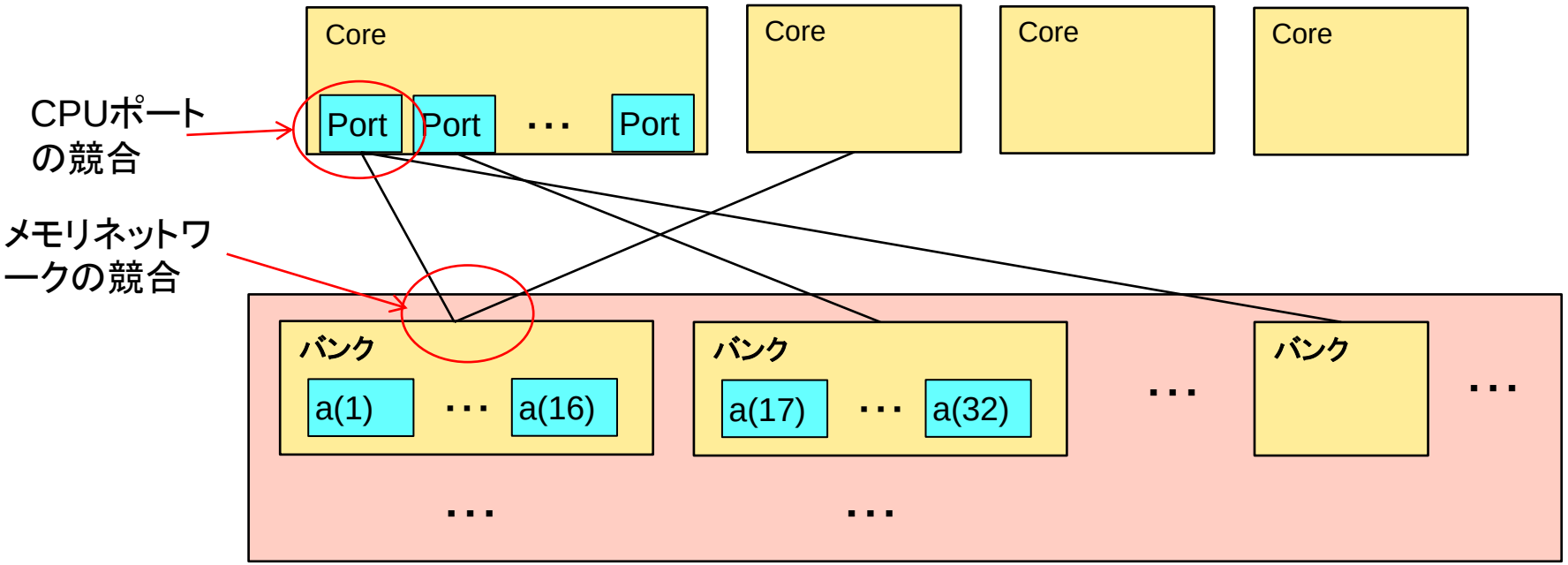
SX-ACEではループ長が10以上であれば連続アクセスとした方が実効性能が高い

(参考：SX-9ではループ長70程度が分岐点)

メモリアクセスの効率化

SX-ACEのメモリは高速化のため128個のバンクから構成される。

- 別々のバンクに対して並列にロード・ストアが可能
- Coreあたり16個のCPUポートを持ち，倍精度浮動小数点型の場合1ポートあたり16個の配列要素を処理



a(i):倍精度浮動小数点型の配列とすると，ひとつのバンクに16要素ずつ格納

バンク競合

複数のデータ転送が同一のバンク・同一のパスを使用するなどの要因により，データ転送性能が低下する現象

(a) CPUポート競合

Core内の同一ポートにロード・ストアの要求が集中したときに発生する

（例）

```
real a(256,10000),b(256,10000)
          :
do i = 1,10000
   a(1, i) = b(1, i)
endo do
```

配列a,bの1次元目の要素数が256である．
配列の1次元目の添字式がループ内で不変であり，2次元目の添字式がループの繰り返しにしたがって1ずつ増加しているので，要素間距離256の等間隔ベクトルとなり，CPUポート競合が発生する．

(b) メモリネットワーク競合

以下のようなケースで発生する

- CPUポート競合が発生しているケース
- 自動並列，OpenMP並列化されたプログラムの複数のタスクから同時に同一バンクにアクセスしたケース．同一の配列要素，スカラ変数を複数のタスクで参照しているときなどに発生する
- 別のプロセスが，たまたま同一のバンクに繰返しアクセスしていたケース

ADBの利用によるメモリ負荷軽減 (1/3)

- ADBを効果的に利用し，メモリアクセス負荷の軽減を図る．
- コンパイラはADBを利用することで高速化が見込める配列に対して自動的にON_ADB指示行を指定するほか，ベクトル版数学関数や行列積ライブラリなどの組込み関数においてADBを使用した高速化を行っている．

```
  7: +------>          do j=1,m
  8: |V----->            do i=1,n
  9: ||      A             d(i,j)=a(i,j)*b(j) + a(i,j-1)*c(j)
 10: |V-----              enddo
 11: +------            enddo

  8  vec   (   1): Vectorized loop.
  8  vec   (  29): ADB is used for array.: a
```

配列aの2次元目の添字式がjとj-1であるので，2次元目のループに繰返しで同じ配列aの要素が参照される（再利用性あり）ので，コンパイラはADBに配列aを乗せることを指定する．

- ADBの容量(1MByte/コア)より，指定する配列が大きい場合は，後からバッファリングするデータにより上書きされる．
- ON_ADB指示行やNOON_ADB指示行を用いて，再利用性の高いデータを選択することでADB利用の効果を引き出す．

ADBの利用によるメモリ負荷軽減 (2/3)

NOON_ADB指示行で指定された配列はADBを利用しない.

```
32: +------>           DO K=2,NZ-2
33: |+----->           DO J=2,NY-2
34: ||           !CDIR NOON_ADB(B1,B2,B3,B4)
35: ||V---->           DO I=2,NX-2
36: |||      A         D= B1(I,J,K)*(A(I+1,J,  K)  +A(I-1,J,  K))
37: |||                &  +B2(I,J,K)*(A(I,  J+1,K)  +A(I,  J-1,K))
38: |||                &  +B3(I,J,K)*(A(I,  J,  K+1)+A(I,  J,  K-1))
39: |||                &  +B4(I,J,K)*(A(I,  J+1,K+1)-A(I,  J+1,K-1)
40: |||                &              -A(I,  J-1,K+1)+A(I,  J-1,K-1))
41: |||      A          C(I,J,K)=A(I,J,K)*D
42: ||V----             END DO
43: |+-----             END DO
44: +------             END DO

35  vec   (   1): Vectorized loop.
35  vec   (  29): ADB is used for array.: a
```

コンパイラは自動で，配列Aだけでなく B1,B2,B3,B4もADBの利用をしようとしている.
ADBの容量，再利用性を考慮し，配列Aだけ利用するため，NOON_ADB指示行で，B1,B2,B3,B4を対象外にする.

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.%	PROC.NAME
1000	11.602(97.5)	11.602	42818.4	19250.5	99.55	255.2	11.602	0.000	0.000	0.000	5.235	20.05	指示行なし
1000	8.948(96.8)	8.948	55519.5	24960.8	99.55	255.2	8.947	0.000	0.000	0.000	3.138	53.27	指示行あり

上記例では，ヒット率が約2.6倍向上し，性能は約1.3倍向上.

ADBの利用によるメモリ負荷軽減 (3/3)

- コンパイラは最適化レベル-Cvoptや-Choptにおいて，ベクトル化されるループ中の配列を自動的に選別し，ADBにバッファリングすることを指定．
- SX-ACEのコンパイラから，バッファリングする配列の種別の選択がオプションにより可能(以下はFORTRAN90/SX)．

形式:
　　-pvctl on_adb [=*category*[:*category*[:...]]]
動作:
　　以下のカテゴリごとにコンパイラがベクトルデータを自動的に ADB に
　　バッファリングする [しない] を制御できる．（*categoryで下のカテゴリ名を指定*）

[no] reuse	再利用性のあるベクトルデータ
[no] indirect	間接指標ベクトル (リストベクトル)
[no] stride	等間隔ベクトル
[no] work	コンパイラが生成する作業配列，作業ベクトルなど
[no] arg	実引数/仮引数
[no] common	共通ブロック
[no] module	モジュール変数
[no] contiguous	連続ベクトル

注意:
　　＊ categoryの指定を省略したとき，以下が指定されたものとみなす．
　　-pvctl on_adb=reuse:indirect:stride:work:arg:common:module

- ON_ADB指示行を指定した手続きに対しては，noreuseとなる．

演習問題2

P77のプログラム例(差分式)を用いて、ADBの効果を検証します．

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 TUNE/practice_2

% ./comp.sh

% qsub run.sh

- 実行が終了すると，run.sh.e.XXXX(標準エラー出力，XXXXはジョブID)ファイルとrun.sh.o.XXXX(標準出力)ファイルが作成されます．catコマンドを用いて標準エラー出力を確認してください．(このプログラムは標準出力には何も出力されません)

% cat run.sh.e.XXXX (プログラム情報の表示)

演習問題2(つづき)

② **コンパイルメッセージ**(sample2.L**をcat)でコンパイラがどの配列を**ADB**に乗せているか確認します.**

% cat sample2.L

- **エディタを用いて**sample2.f**を編集し,** ON_ADB/NOON_ADB**指示行を挿入してADBに乗せる配列を選択して,コンパイルとジョブ投入を行ってください.挿入する位置は下記リストに示す位置です.**

```
32: +------>          DO K=2,NZ-2
33: |+------>         DO J=2,NY-2
（ここに指示行挿入）
35: ||V---->          DO I=2,NX-2
36: |||               D= B1(I,J,K)*(A(I+1,J,  K)  +A(I-1,J,  K))
37: |||              &  +B2(I,J,K)*(A(I,  J+1,K)  +A(I,  J-1,K))
38: |||              &  +B3(I,J,K)*(A(I,  J,  K+1)+A(I,  J,  K-1))
39: |||              &  +B4(I,J,K)*(A(I,  J+1,K+1)-A(I,  J+1,K-1)
40: |||              &              -A(I,  J-1,K+1)+A(I,  J-1,K-1))
41: |||               C(I,J,K)=A(I,J,K)*D
42: ||V----           END DO
43: |+-----           END DO
44: +------           END DO
```

- **編集が終了しましたら,再度コンパイル・ジョブ投入を行ってください.**

MPIプログラムの分析 (1/3)

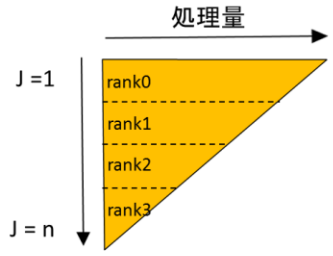
並列プログラムの性能を分析する場合，転送時間がコストの上位を占める場合，データ転送処理のコストであるか，演算のインバランスを吸収している結果であるか見極める必要がある．

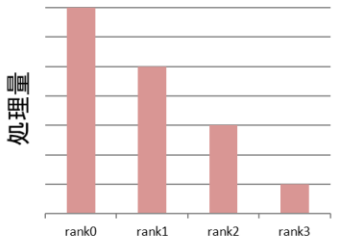
```
31: +------>          do it=1,m
32: |                  sum1=0.0d0
33: |                  call sub1(it)
34: |                  call sub2
35: |                  sum3=sum3+sum2
36: +------           enddo

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 (1/2)

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

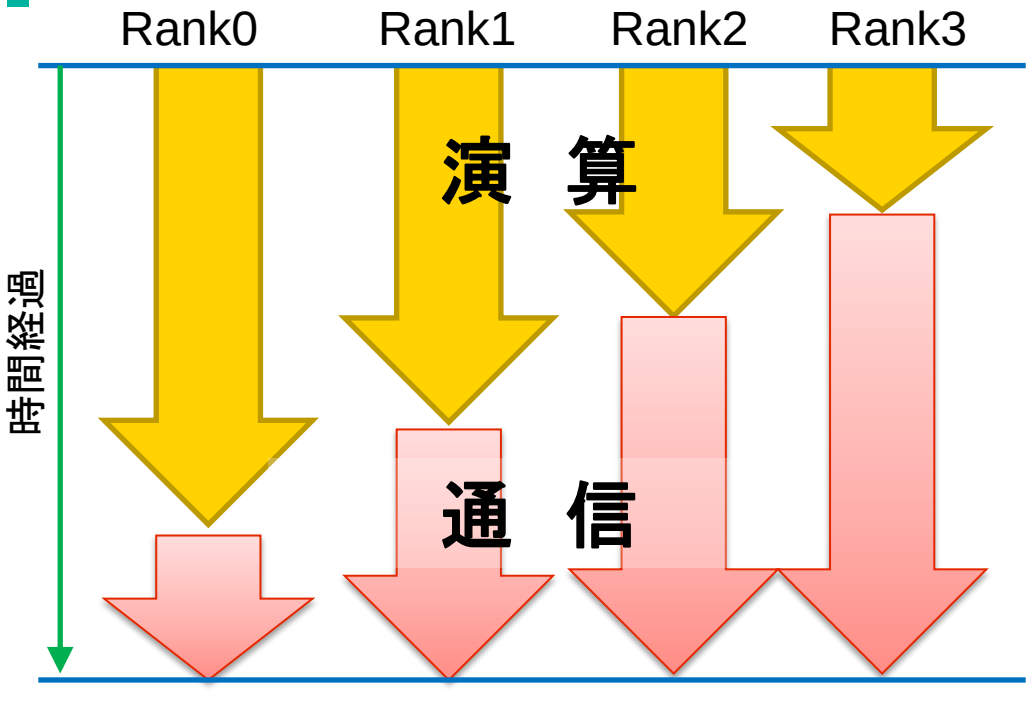
演算を行っているsub1のコスト(57%)に対して，MPI_ALLREDUCEを行っているsub2のコストが34%と比較的大きい．これはプロセスの待ち時間がMPI_ALLREDUCEにカウントされるためである．

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 (1/3)

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.

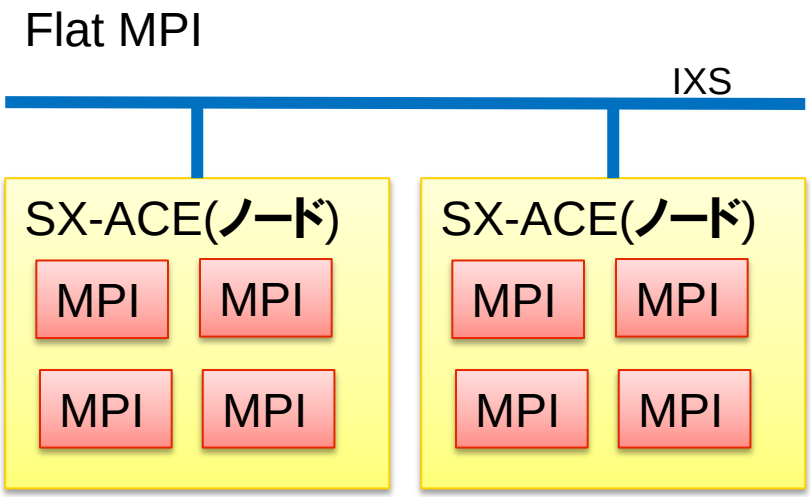
sub2をcallする前にMPI_BARRIERで同期を取ることで，sub2にプロセスの待ち合わせ時間を含めない.

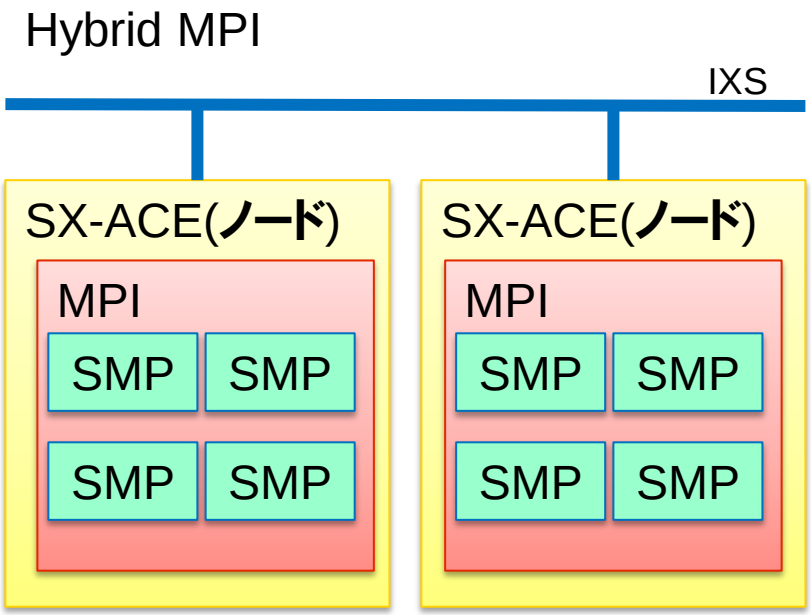
PROC.NAME	ELAPSED TIME[sec]	COMM.TIME [sec]	COMM.TIME / ELAPSED	IDLE TIME [sec]	IDLE TIME / ELAPSED	AVER.LEN [byte]	COUNT	TOTAL LEN [byte]
sub2	0.020	0.018		0.004		10.0	8000	78.1K
0.0	0.020	0.018	0.925	0.003	0.155	16.0	2000	31.2K
0.1	0.017	0.016	0.927	0.004	0.209	8.0	2000	15.6K
0.2	0.006	0.005	0.791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

ハイブリッド実行 (1/4)

SX-ACEの複数ノードを利用のプログラム実行には，次の二つの並列実行が可能である

- すべてのコアを用いてMPI並列実行する(Flat MPI)
- ノード内は自動並列(もしくはOpenMP並列)を利用し，ノード間をMPI並列実行するハイブリッド並列(Hybrid MPI)





SMP:自動並列(もしくはOpenMP並列)

ハイブリッド実行 (2/4)

Flat MPIとHybrid MPIの比較

	特長	選択のポイント
Flat MPI	すべてのコアが通信処理を担当	転送長が小さい隣接間通信が多い場合
Hybrid MPI	領域を大きく分割でき、転送回数を削減	MPI_ALLGATHERに代表される全プロセス間を通信する転送長が大きい集団通信が多い場合

隣接間通信の例

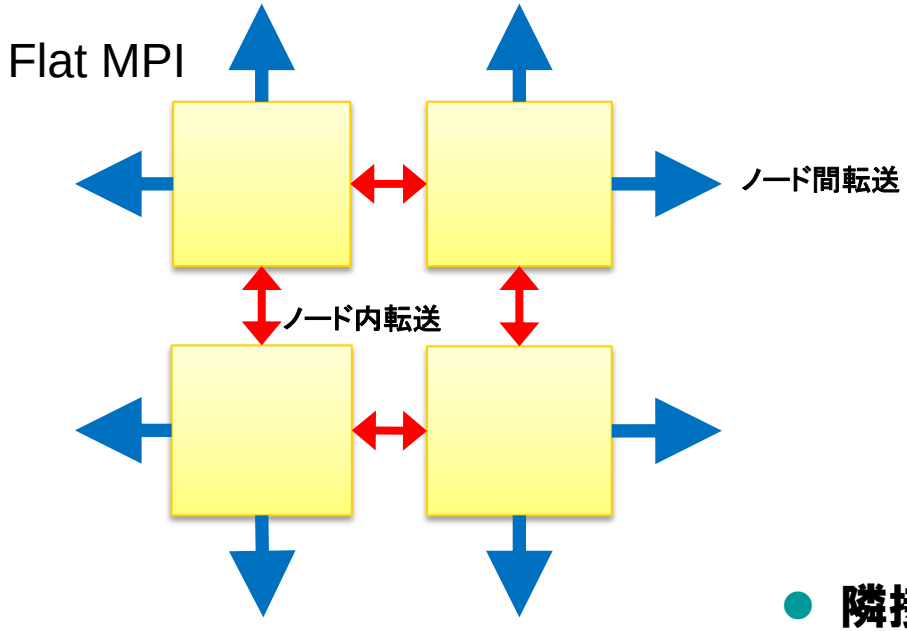
- 差分式を有する場合は，隣接プロセスが担当する領域のデータを参照する必要があり，一対一通信で転送する必要がある．

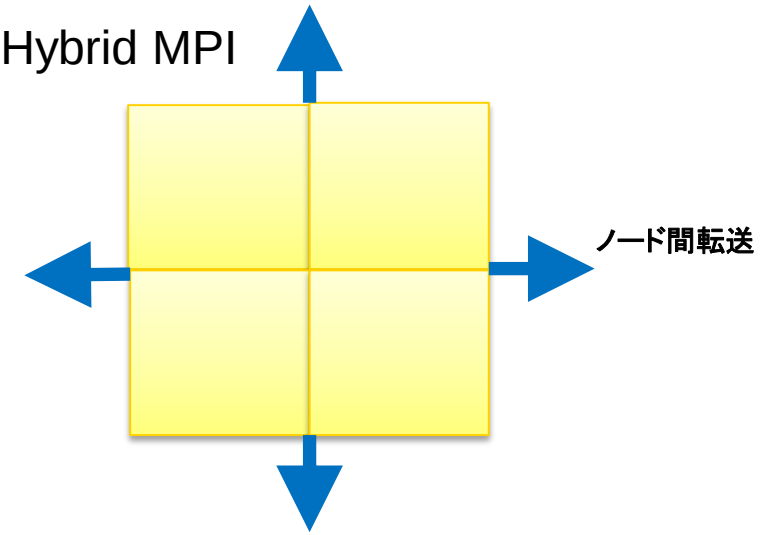
```
 7: +------>          do j=ist,ied
 8: |V----->            do i=1,n
 9: ||      A             d(i,j)=a(i,j)*b(j) + a(i,j-1)*c(j)
10: |V-----              enddo
11: +------            enddo
```

左記の例では，配列aはj-1の要素を参照するので，更新されるタイミングで隣接プロセスからデータを転送する必要がある．

ハイブリッド実行 (3/4)

隣接間通信の例(つづき)

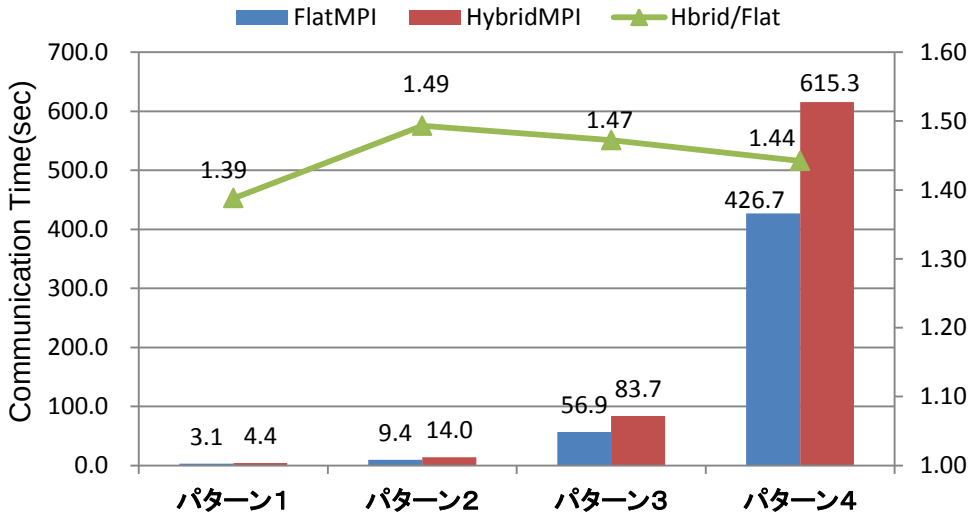




- 一回の転送長

	FlatMPI	HybridMPI
パターン1	2.3KB	4.5KB
パターン2	18.0KB	36.0KB
パターン3	144.0KB	288.0KB
パターン4	1.1MB	2.3MB

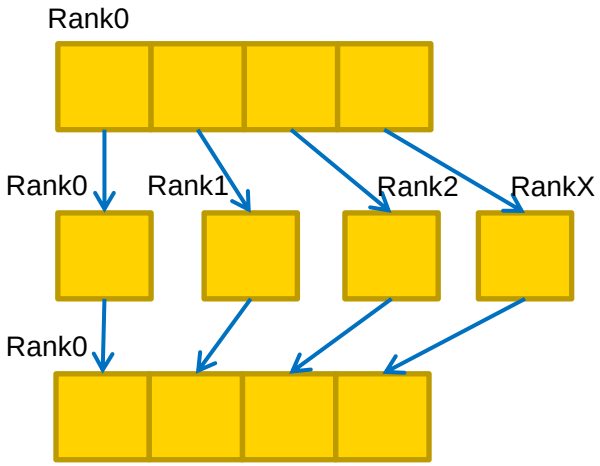
- 隣接間転送を繰り返し実行した評価結果

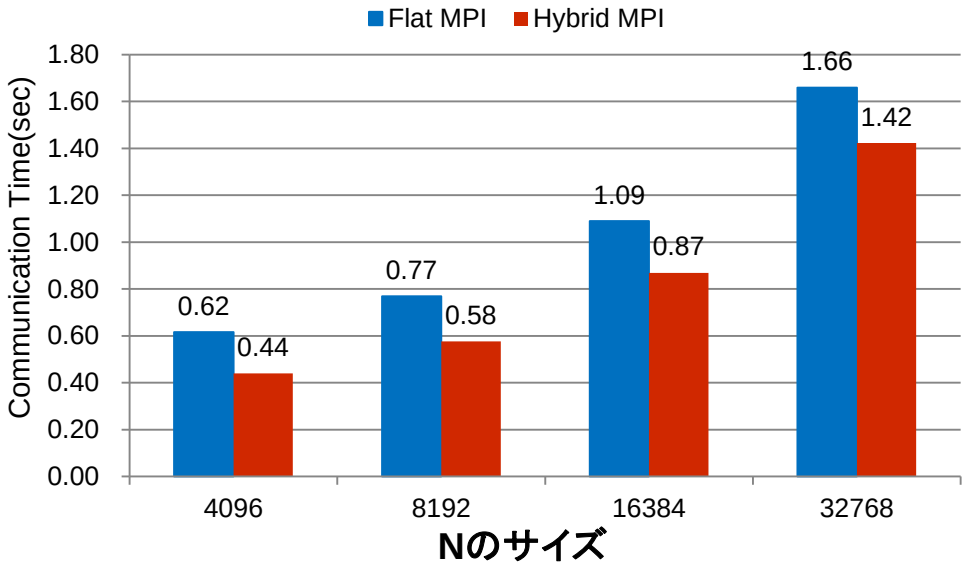


ハイブリッド実行 (4/4)

集団通信(MPI_SCATTER/MPI_GATHER)の例

- 倍精度実数型の配列A(N,N)を全プロセスに配り(MPI_SCATTER)，全プロセスから集める(MPI_GATHER)転送処理を10,000回繰り返す実験
- Flat MPI(ノード数×コア数で分割)とHybrid MPI(ノード数で分割)をSX-ACEの4ノードで実行





- 一回の転送長

N	FlatMPI	HybridMPI
4,096	8MB	32MB
8,192	32MB	128MB
16,384	128MB	512MB
32,768	512MB	2GB

- 一回当たりの転送量はFlat MPIの方が小さいが，ノード当たりの通信回数を削減するHybrid MPIの方が転送時間は短い．
- MPI_ALLGATHERやMPI_ALLTOALLの全対全通信ではさらに差は大きくなる．

5. 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**(http://www.ss.cc.tohoku.ac.jp/report/speed-up.html)

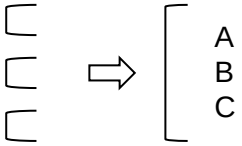
[補足資料]

ARRAYCOMB指示行(1/2)

- **配列式は内部的に，一文が多重ループに展開される．引き続く配列式の文は，それぞれループになる．同型のループになれば自動的に融合される．**
- **融合できるか否かがコンパイラに不明な場合でも**ARRAYCOMB **と**END ARRAYCOMB **の間にある配列式は，(強制的に)ループ融合する．なおこの機能は、以前からあった指示行．**

```
SUBROUTINE SUB2(N)
INTEGER,DIMENSION(1:N)::A,B,C
A(:)=1
B(:)=2
C(:)=3
```

指示行不要。融合してベクトル化




```
f90: opt(11): array.f90, line 3: 配列代入を融合した．:line 3 - 5
f90: vec(4): array.f90, line 3: 配列式全体をベクトル化する．
```

```
SUBROUTINE SUB(I,J,K)
REAL,ALLOCATABLE,DIMENSION(:)::A,B,C
ALLOCATE(A(1:I))
ALLOCATE(B(1:J))
ALLOCATE(C(1:K))
!CDIR ARRAYCOMB
A(:)=1
B(:)=2
C(:)=3
!CDIR END ARRAYCOMB
END
```

ループ融合してベクトル化される（I=J=Kと仮定）
指示行がないと，3つのベクトルループになる．

ARRAYCOMB指示行(2/2)

SX-ACEのコンパイラで，ARRAYCOMBの後ろに，別の指示オプションが指定可能．

```
SUBROUTINE SUB(I,J)
    REAL(KIND=4),ALLOCATABLE,DIMENSION(:,:)::A,B
    ALLOCATE(A(1:I,1:I),B(1:J,1:J))
!CDIR ARRAYCOMB,COLLAPSE
    A(:,:) = 1.0          !(1)
    B(:,:) = 2.0          !(2)
!CDIR END ARRAYCOMB
END SUBROUTINE SUB
```

(1)と(2)が融合され，融合されたループは一重化される(I=Jを仮定)

DATA_PREFETCH(変数|配列|配列要素[,要素数])指示行

- 指定の変数，配列，配列要素を先頭として，要素数分ADBにプリフェッチする．
- 要素数が省略された場合は，変数や配列要素の場合はその1個，配列名の場合はその配列全体の指定となる．
- VPREFETCH指示行との違いは，VPREFETCH指示行がループ内の指定配列の連続・等間隔ベクトルロードに対して繰り返し毎に指定回数(既定値3)前の繰り返しでプリフェッチするように調整されたアドレスを使用するのに対して，DATA_PREFETCH指示行はその場で指定のアドレスからデータをプリフェッチするものである．

```
!CDIR DATA_PREFETCH(A(1),1000)
!CDIR DATA_PREFETCH(B(1),1000)
ループ前にA,Bの1～1000要素をプリフェッチする
 …
!CDIR ON_ADB(A)
!CDIR ON_ADB(B)
DO I=1,1000000
X(I) = A(IDX(I)) + B(IDX(I)) ! リストベクトルのロード
Y(I) = A(IDX(I)) - B(IDX(I)) ! リストベクトルのロード
...
ENDDO
```

```
DO J=1,10000
!CDIR VPREFETCH(A,3)
!CDIR VPREFETCH(B,3)
DO I=1,10240 ! ベクトル化されたループ
... ループの各繰返しでプリフェッチが出る
    要素数は256＊2(AとB)[＊アンロール数]
... = A(I,J) +B(I)
...
END DO
END DO
```

*ループ前にも充填のためプリフェッチが出る

- DATA_PREFETCH指示行は，ADBに載せる配列か小さい場合や，特にリストベクトルのような非連続アクセスの場合にロードの高速化が期待できる．

TRACEBACK指示行

TRACEBACK 指示行を指定した位置に実行が到達したとき，標準エラー出力にメッセージとトレースバック情報を出力することを指定する．

```
      program p
      real a(100)
      call sub1(a,100,s)
      write(6,*) s
      end
      subroutine sub1(a,n,s)
      real a(n)
      do i=1,n
       a(i) = i
      enddo
      call sub2(a,n,s)
      end
      subroutine sub2(a,n,s)
      real a(n)
      s = 0
!cdir traceback
      do i=1,n
      s = s + a(i)
      enddo
!cdir traceback
      end
```

```
$ f90 -Wf"-dir debug" traceback.f90
$ ./a.out
 * Trace-back information PROG=sub2 ELN=16(40000153c)
                Called from sub1 ELN=11(4000011c4)
                Called from p ELN=3(400000a50)
 * Trace-back information PROG=sub2 ELN=20(40000156c)
                Called from sub1 ELN=11(4000011c4)
                Called from p ELN=3(400000a50)
 5050.000
$
```

- -Cdebug以外の場合，コンパイル時詳細オプション-dir debugが必要
- 情報出力後も，実行は継続する