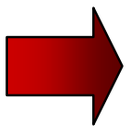
HPF/ESによるNPBの並列化

地球シミュレータセンター
村井 均

はじめに

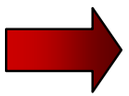
- 昨年11月にリリースされたNPB3.0alphaにはHPF版が含まれる。
- バグがある上に性能が悪い。特にベクトル機では非常に悪い。



- ✓ HPF/ESによるNPBの高効率な実装を求める。
- ✓ ノウハウを蓄積するとともに、課題を抽出する。

NPB3.0alpha

- *FT(三次元FFT)* → そのままでMPI版(NPB2.4)に匹敵する。
- *IS(整数ソート)* → Fortran版が提供されていないので保留。
- *EP(並列乱数)* → NPB3.0alphaには含まれないが、容易に並列化できる。



残る5つ*(BT, SP, LU, CG, MG)*をHPF/ESで並列化する。

方針

MPIコードを元に書かれているため、逐次コードとしては効率が悪い。

- 最新のシリアル版(NPB2.3-serial)をベースとする。
- ベクトル化を目的とした(
チューニングを行う。
- ソースの書き換えも行うが、基本的には指示文の追加のみ。
- MPI版(NPB2.4)と同じ処理(アルゴリズム)を目指す。
- 一部を除いて一次元BLOCK分散を用いる。

NPB3.0alphaの書き換えより少ない。

BTベンチマーク

「ADI法による5×5ブロック三重対角行列ソルバ」

- ✓ HPF並列化のポイント
  - 引数の再マッピング
  - 並列callとインライン展開
  - 分散テンポラリ

引数の再マッピング

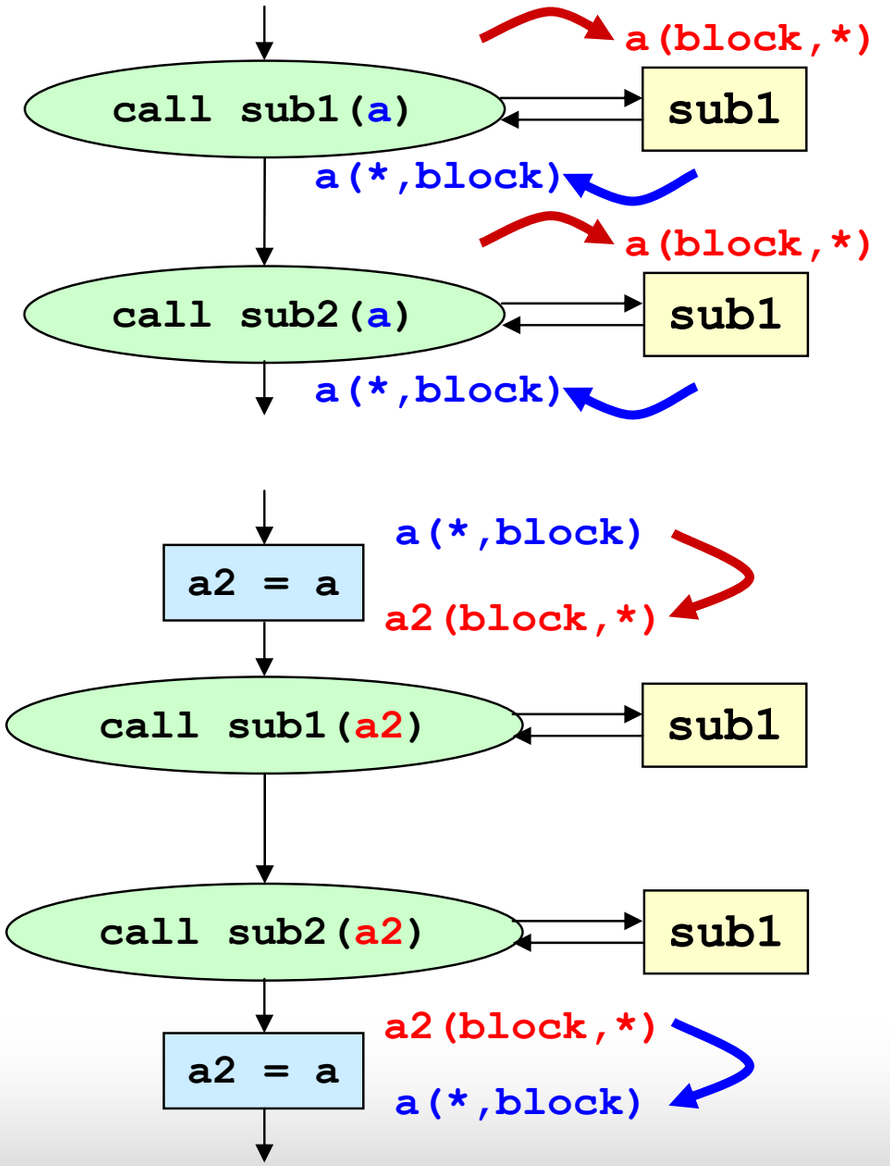
引数の再マッピングが起きる場合

```
!HPF$ distribute (*,block) :: a
      call sub1(a)
      call sub2(a)
```

コピーを明示的に書くことで再マッピングの回数を減らす。

```
!HPF$ distribute (*,block) :: a
!HPF$ distribute (block,*) :: a2
      a2 = a
      call sub1(a2)
      call sub2(a2)
      a = a2
```

※ aはcommonなのでredistributeは不可



並列callとインライン展開

```
      do k=1,N3
        do j=1,N2
          call sub(a(1,1,bb,0,j,k))
        end do
      end do
```

```
!HPF$ sequence t

      interface
        pure extrinsic(FORTRAN_LOCAL)
     +  subroutine sub(t)
        ...
      end interface

!HPF$ independent
      do k=1,N3
        do j=1,N2
          t(:,:) = a(:,:,bb,0,j,k)
          call sub(t(1,1))
          a(:,:,bb,0,j,k) = t(:,:)
        end do
      end do
```

jループをベクトル化

```
!HPF$ independent
      do k=1,N3
        t(:,:,:) = a(:,:,bb,0,:,k)
        do j=1,N2
          call sub(t(1,1,j))
        end do
        a(:,:,bb,0,:,k) = t(:,:,:)
      end do
```

- インタフェースブロックでpure属性を宣言し、並列ループ中からcallする。
- コンパイラオプションの指定によって、インライン展開する。

```
-pi exp=sub expin=...
```

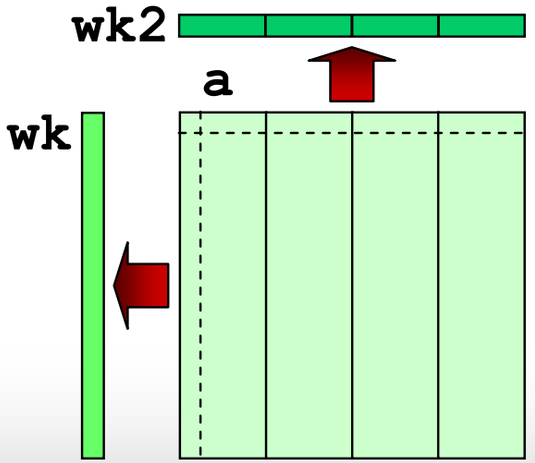
ベクトル化

分散テンポラリ

```
      real a(n,n), b(n,n), wk(n)
!HPF$ distribute (*,block) :: a

      do j=1, n
        wk(j) = a(1,j)
        b(1,j) = wk(j) + ...
      end do

      do i=1, n
        wk(i) = a(i,1)
        b(i,1) = wk(i) + ...
      end do
```




```
      real a(n,n), b(n,n), wk(n)
!HPF$ distribute (*,block) :: a
      real wk2(n)
!HPF$ distribute (block) :: wk2

      do j=1, n
        wk2(j) = a(1,j)
        b(1,j) = wk2(j) + ...
      end do

!HPF$ on home(b(:,1)), local
      do i=1, n
        wk(i) = a(i,1)
        b(i,1) = wk(i) + ...
      end do
```

テンポラリ配列を使いまわすには、サイズだけでなくマッピングも一致する必要がある。

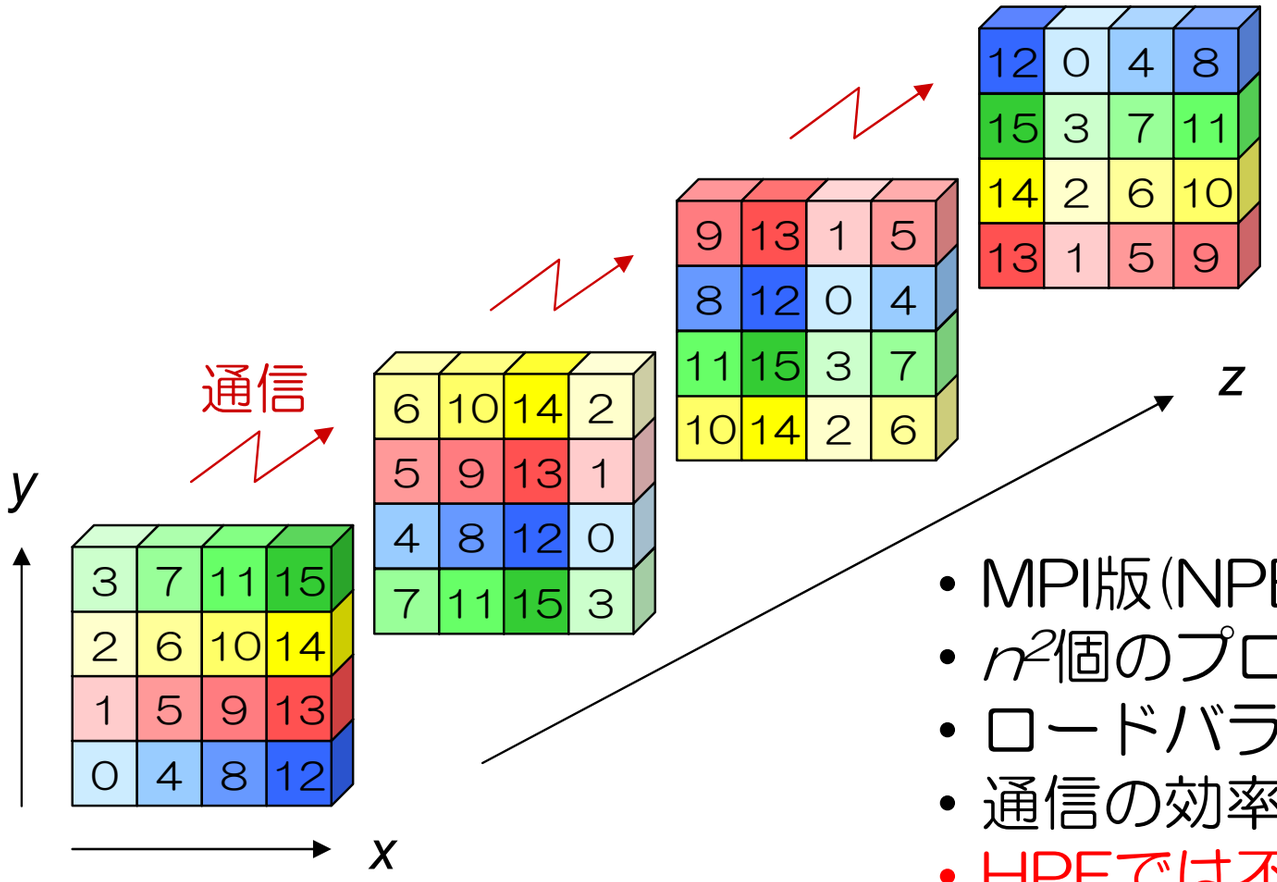
SPベンチマーク

「ADI法によるスカラ5重対角行列ソルバ」

- ✓ HPF並列化のポイント
  - 引数の再マッピング
  - 並列callとインライン展開
  - 分散テンポラリ

→ BTと全く同じ

マルチパーティション



- MPI版(NPB2.4)のBTとSPで採用。
- n^2個のプロセッサ
- ロードバランスに優れる。
- 通信の効率が良い。
- HPFでは不可。
- Rice大学のdHPFコンパイラで実装されている。

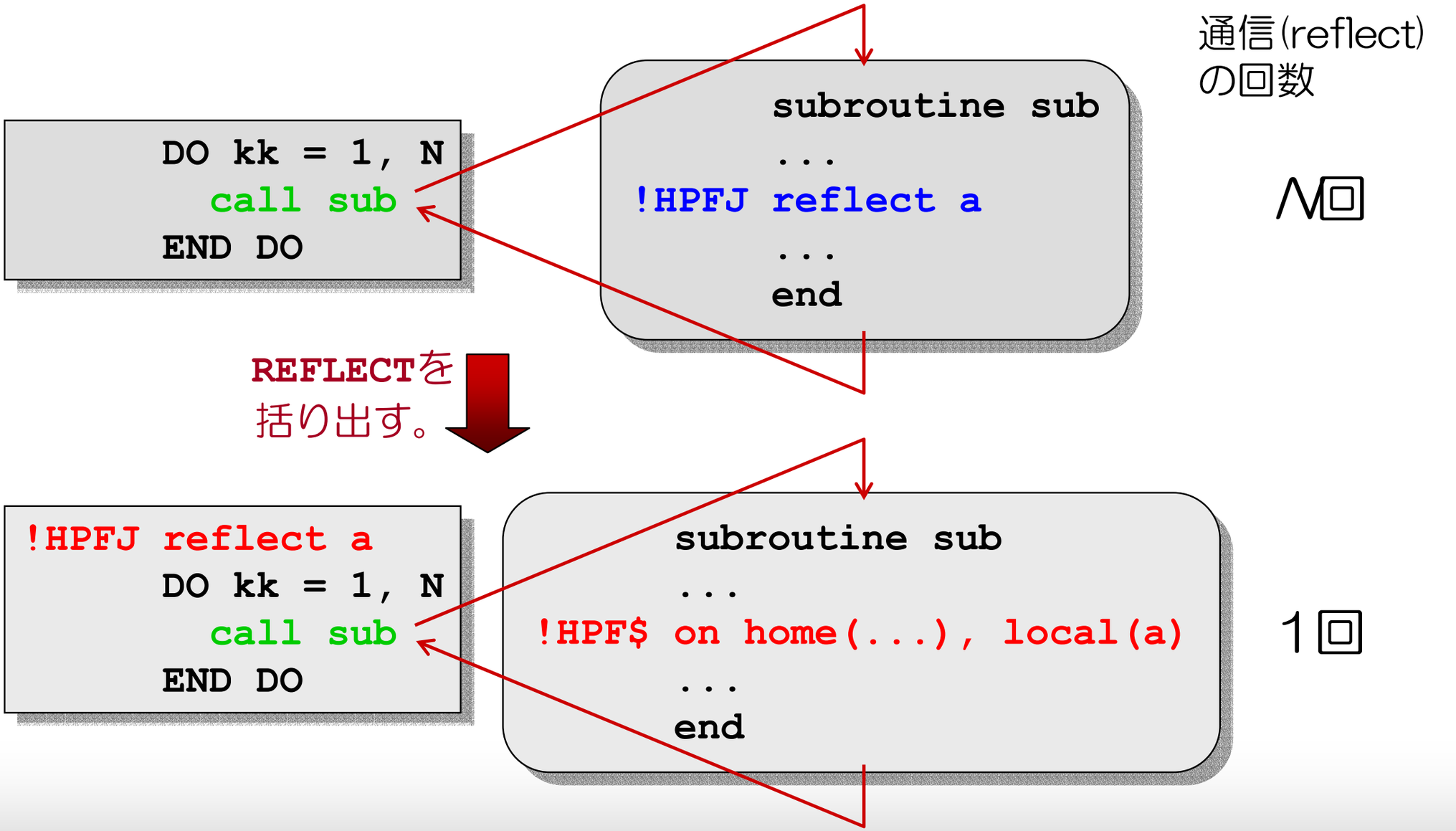
LUベンチマーク

「対称SOR法による5×5ブロック上下三角行列ソルバ」

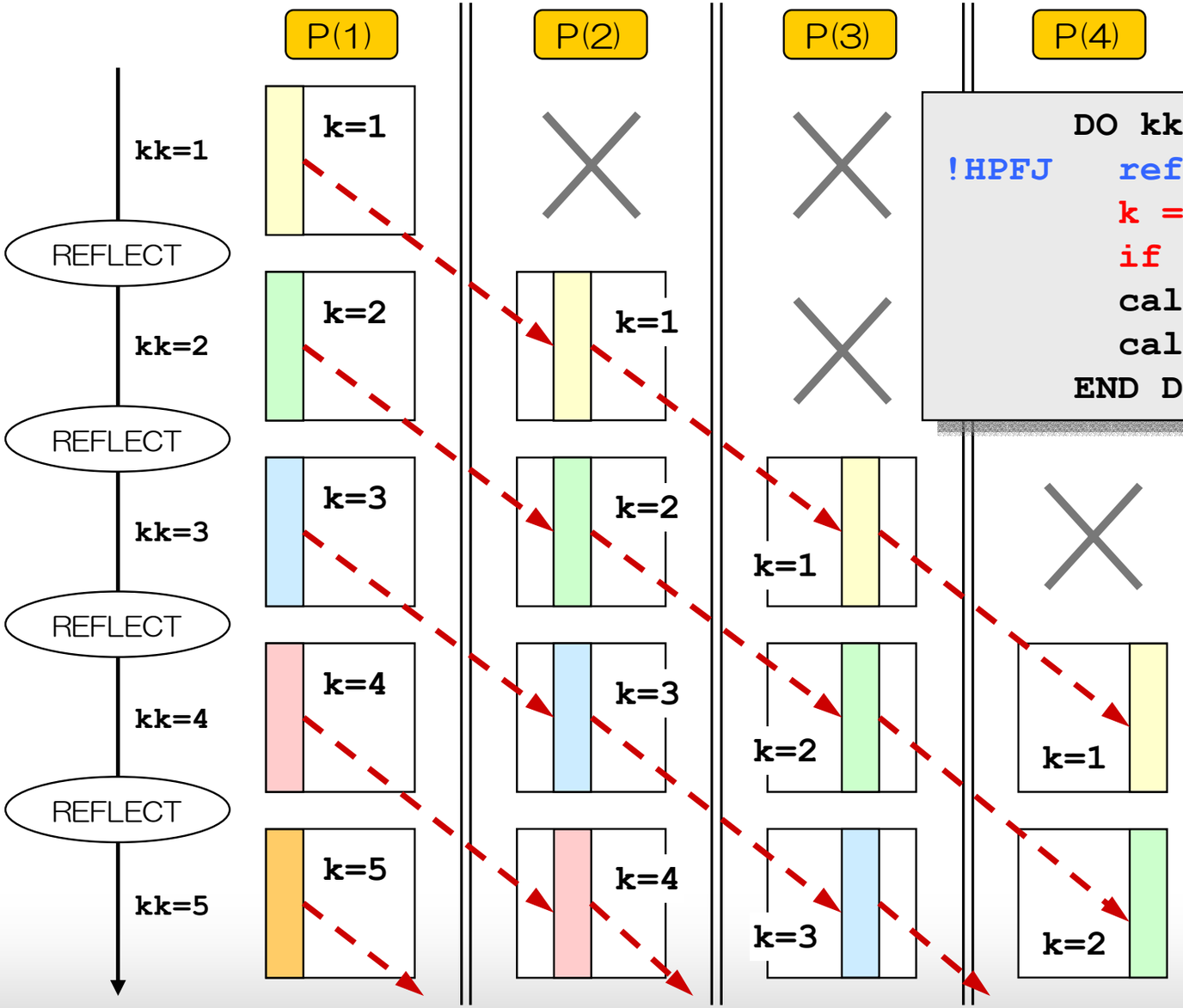
- ✓ HPF並列化のポイント
  - **REFLECT**指示文の括り出し
  - パイプライン実行
  - (**ON EXT_HOME**指示文による重複計算)
  - (部分的REFLECT)

HPF/ESはサポートしないので、ソースまたは中間コードを手で変換して効果を評価した。

REFLECT指示文の括り出し



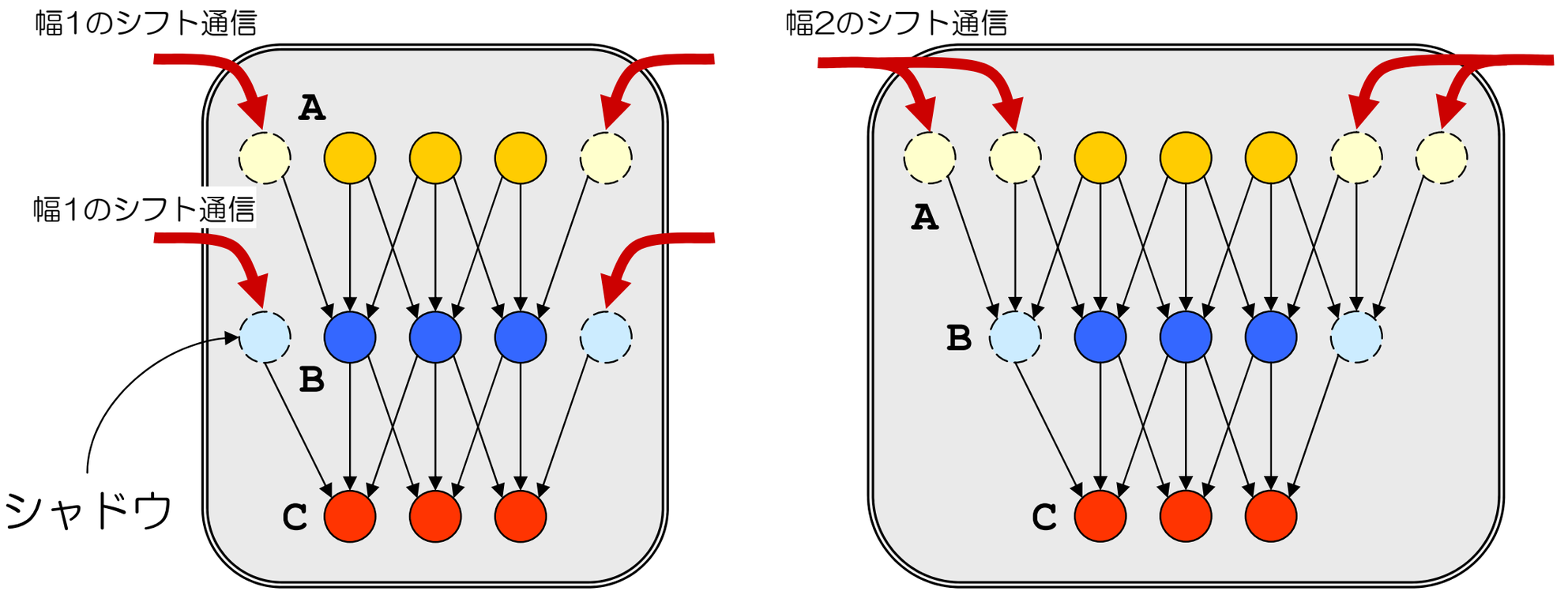
パイプライン実行




```
      DO kk = 1, nz+np-1
!HPFJ   reflect rsd
        k = kk - myid
        if (k < 1 .or. k > nz) cycle
        call jacld(k)
        call blts(…, k, …, rsd, …)
      END DO
```

※ REFLECTの実行によって、全プロセッサの同期がとられるので、完全なパイプライン実行ではない。

EXT_HOMEによる重複計算

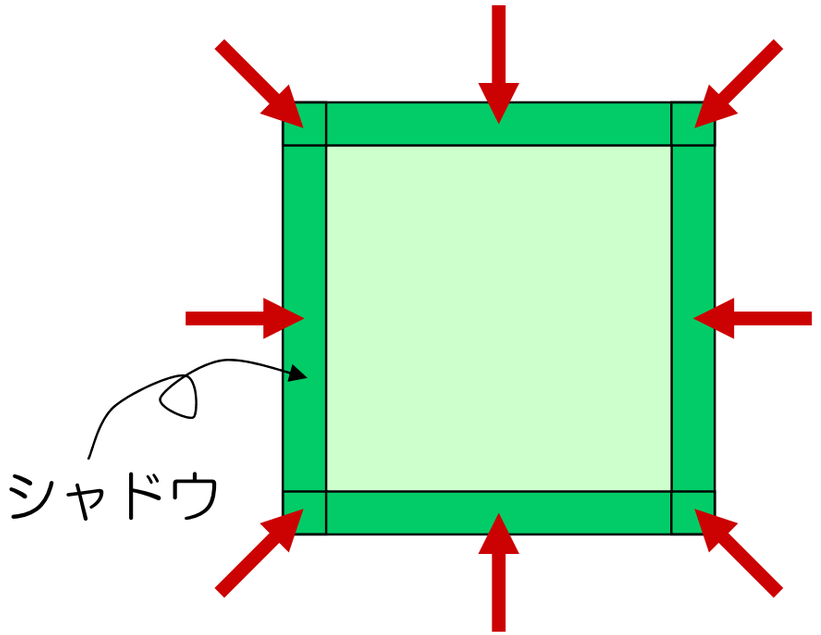


通常のオーナコンピュート
ルールに従う計算

EXT_HOMEによる重複計算

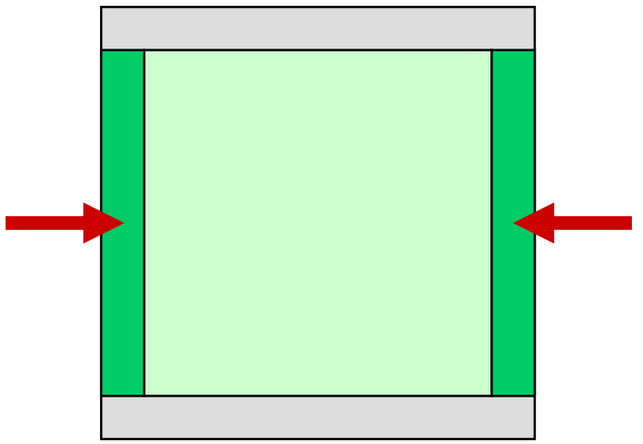
重複計算により、通信の回数を削減できる(一回の通信量は増える)。

部分的REFLECT



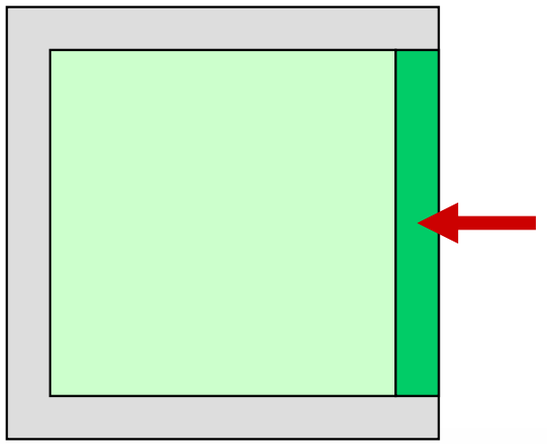
!HPFJ reflect a

HPF/JA仕様の**REFLECT**指示文は、対象の配列の全てのシャドウ領域を更新する。



!HPFJ reflect a(0,1:1)

シャドウ領域の一部だけを更新するように指定したい。



!HPFJ reflect a(0,0:1)

- HPF/JA仕様では不可。
- Rice大学のdHPFコンパイラで実装されている。

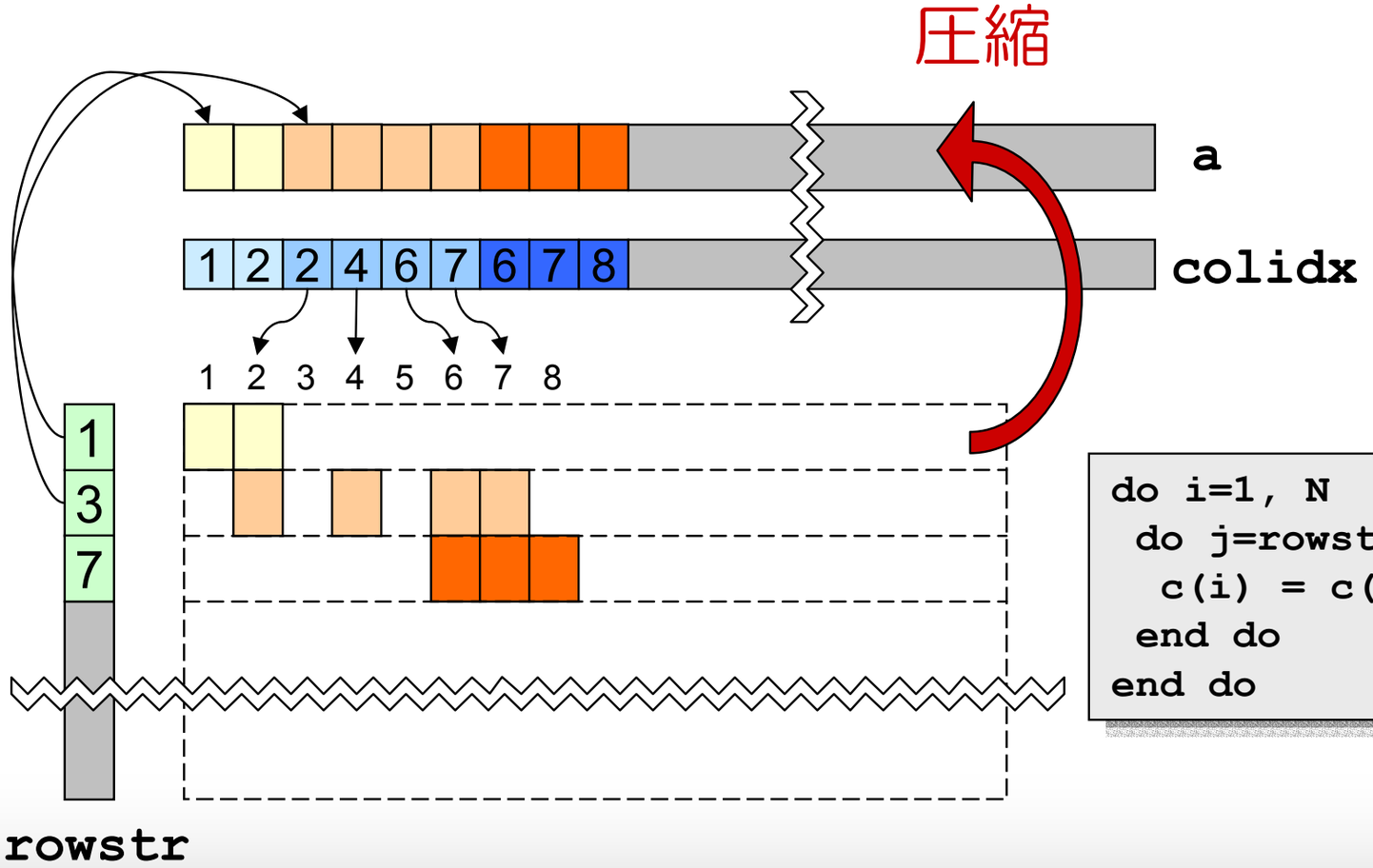
CGベンチマーク

「共役勾配法」

- ✓ HPF並列化のポイント
  - 疎行列データ構造(CRS: Compressed Row Storage)の分散
    - GEN_BLOCK分散による方法
    - 二次元分散とローカルデータによる方法

評価対象

CRS形式で圧縮された疎行列に対する行列ベクトル積

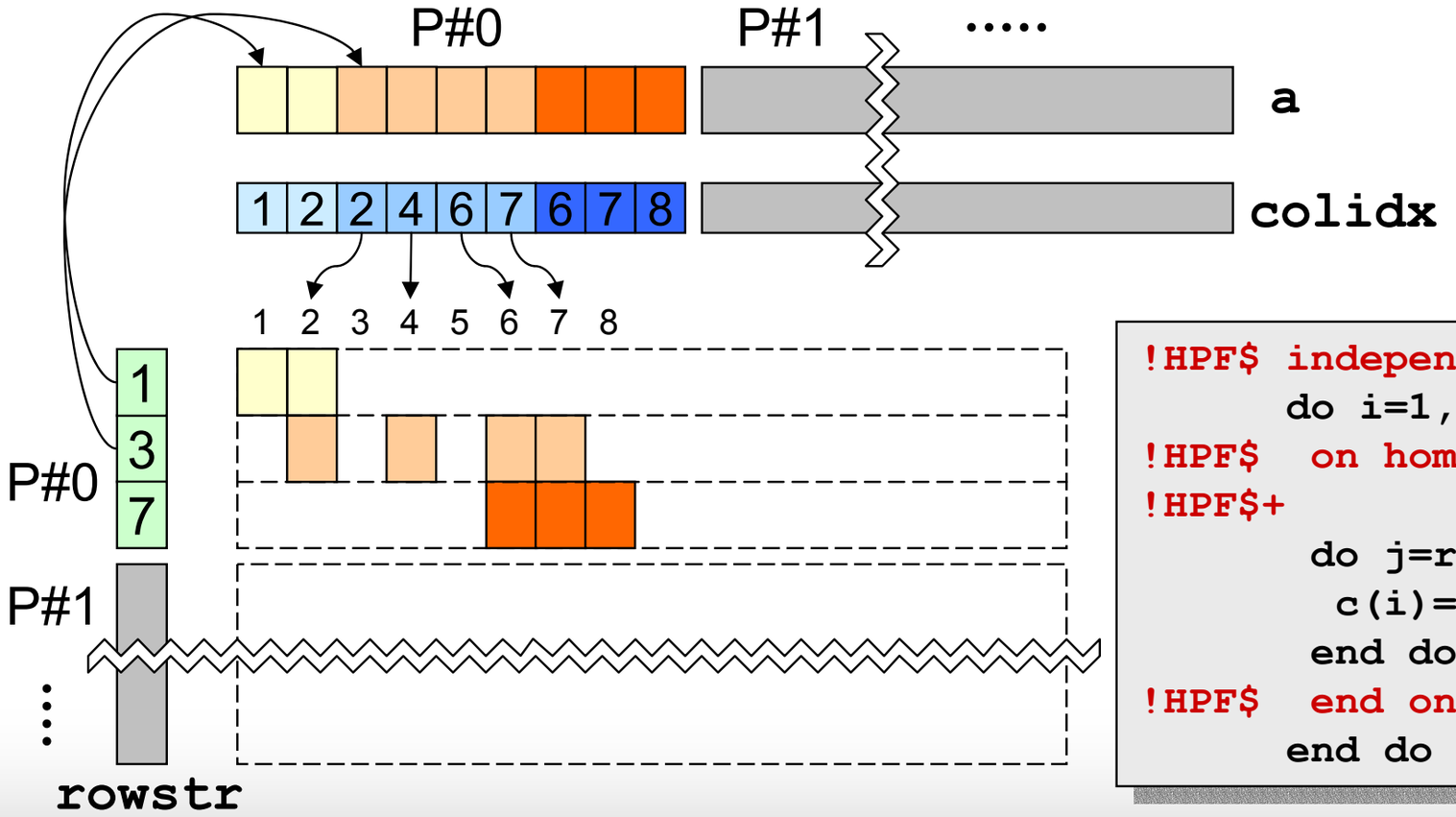



```
do i=1, N
 do j=rowstr(i), rowstr(i+1)-1
  c(i) = c(i) + a(j)*b(colidx(j))
 end do
end do
```

方法(1) GEN_BLOCK

rowstrの分散に合わせて、圧縮された配列**a**のマッピング配列**map**を設定する。

```
      integer :: map(NP) = (/9, .../)
!HPF$ DISTRIBUTE (BLOCK) :: c, rowstr
!HPF$ DISTRIBUTE (GEN_BLOCK(map)) :: a, colidx
```

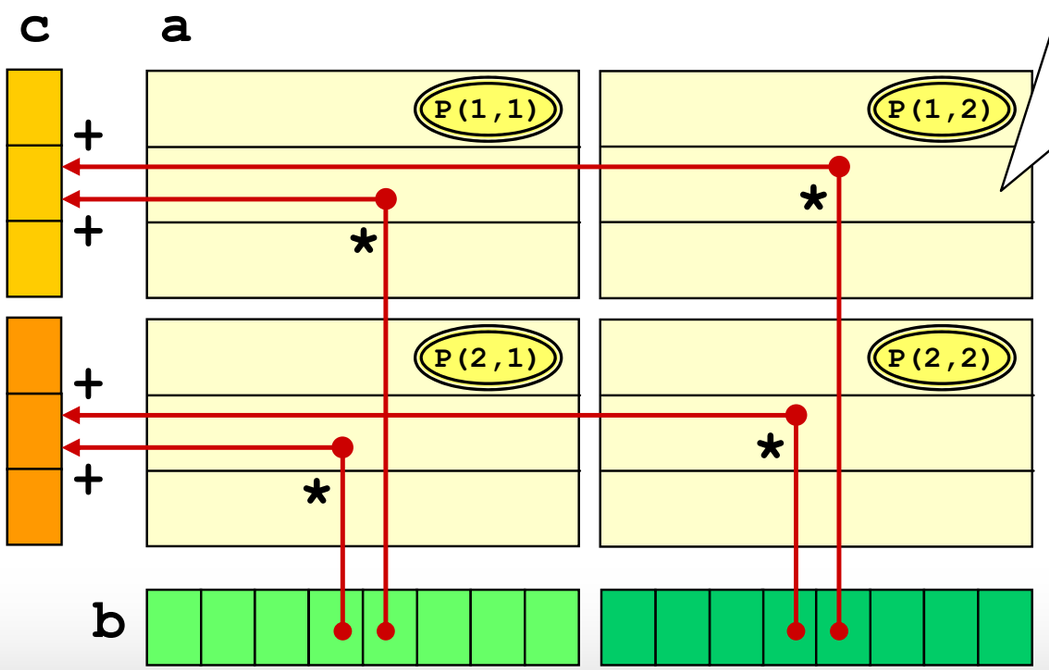


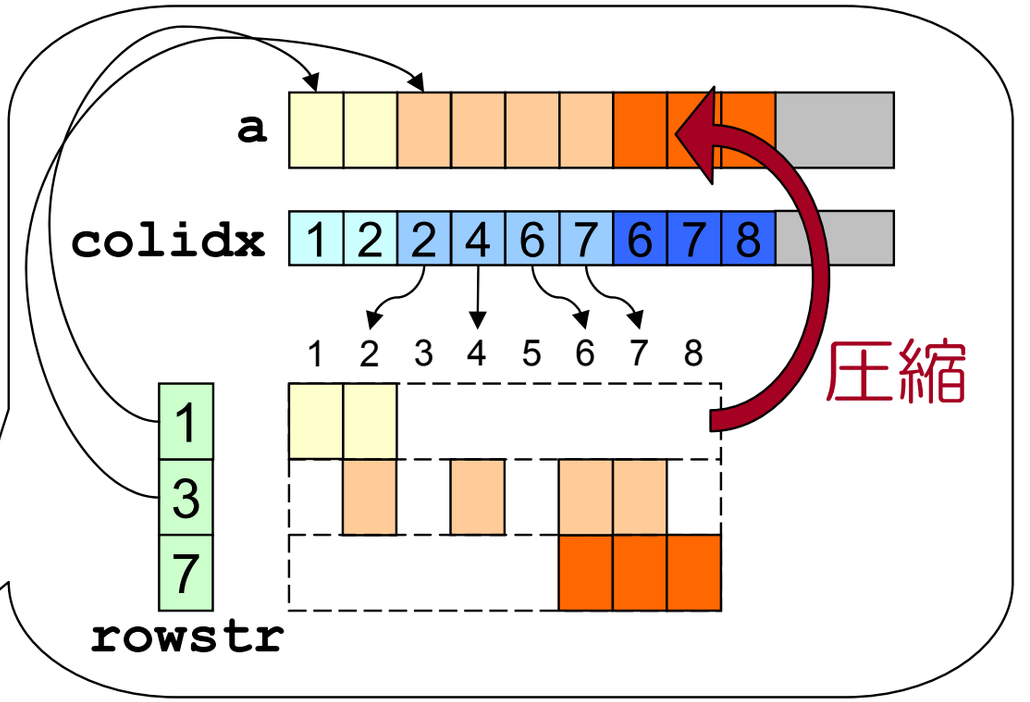

```
!HPF$ independent
      do i=1, N
!HPF$  on home(rowstr(i)),
!HPF$+        local(a, colidx) begin
       do j=rowstr(i),rowstr(i+1)-1
        c(i)=c(i)+a(j)*b(colidx(j))
       end do
!HPF$  end on
      end do
```

方法(2) 二次元BLOCK分散

```
!HPF$ template t(N,N)
!HPF$ distribute t(block,block)
!HPF$ align b(j) with t(*,j)
!HPF$ align c(i) with t(i,*)
```

a,colidx,rowstrは非分散(ローカル)
→ ローカル手続きで設定






```
!HPF$ independent, reduction(c)
      do i=1, N
       do j=rowstr(i), rowstr(i+1)-1
!HPF$   on home(t(i,j)), local(b)
        c(i) = c(i) + a(j)*b(colidx(j))
       end do
      end do
```

MGベンチマーク

「マルチグリッド法」

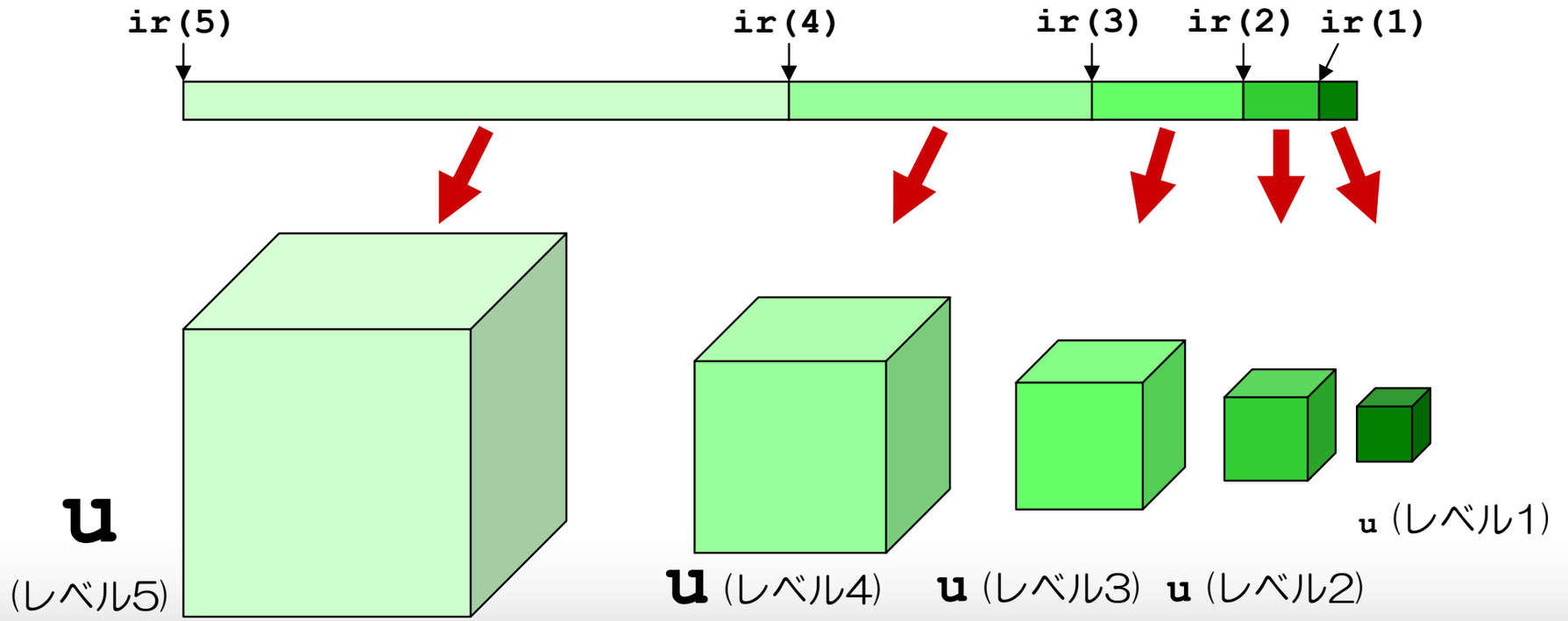
- ✓ HPF並列化のポイント
  - 再帰呼び出しによる階層並列化
  - 階層データのマッピング

NPB2.4における階層並列の実現

```
call psinv(u(ir(k)),...)
...
subroutine psinv(u,...)
real u(n1,n2,n3)
...
```

一次元配列の要素を渡し、三次元配列として受け取る。

➡ HPFでは分散できない。



従来手法(1)

階層ごとに別々の配列と別々の**call**文を用意する[1]。

```
      real u1(n1,n1,n1), u2(n2,n2,n2), u3(n3,n3,n3)
!HPF$ distribute (*,*,block) :: u1, u2, u3
      ...
      if (k == 1) call psinv(u1, ...)
      else if (k == 2) call psinv(u2, ...)
      else if (k == 3) call psinv(u3, ...)
```

階層の数が増えると配列の宣言と**call**文を追加する必要があり、現実的でない(面倒)。

➡ 配列を番号付けして扱うことが必要。

[1] Y. Nishitani et al.: *Techniques for compiling and implementing all NAS parallel bench marks in HPF*, Concurrency and Computation - Practice & Experience - Special Issue : High Performance Fortran, Vol. 14, No. 8-9, Wiley, pp. 769-787, July-10 August 2002.

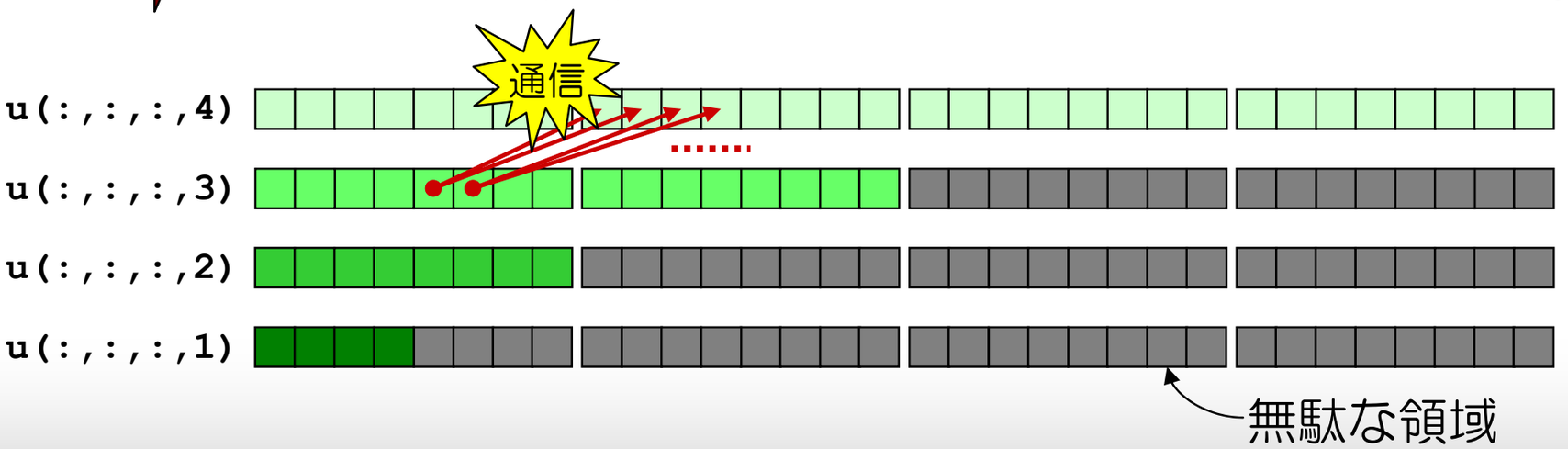
従来手法(2) — NPB3.0

配列の次元を拡張し、レベル毎に割り当てる。

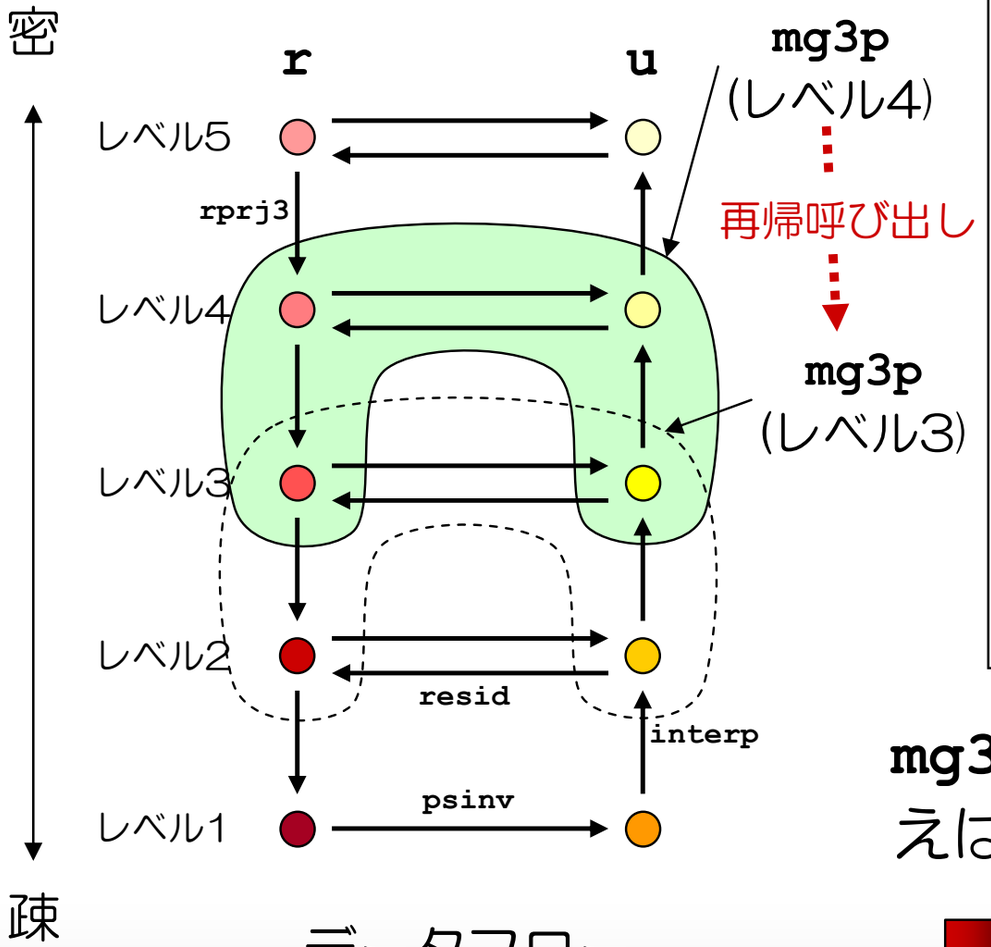
```
      real u(n1,n2,n3,ilevel)
!HPF$ distribute (*,*,block,*)
```

- 無駄な領域が増え、負荷の不均衡が生じる。
- 階層間のコピーで通信が発生する。

➡ 配列の整列関係を正しく指定することが必要。



新しい手法(再帰呼び出し)



データフロー

```
recursive subroutine mg3P(u,r,k)
...
call rprj3(r,r2,k)

if (k > 2) then
  call mg3p(u2,r2,k-1)
else
  call psinv(r,u,1)
end if

call interp(u2,u,k)
call resid(u,r,k)
call psinv(r,u,k)

end
```

mg3Pは、*k*と*k-1*の2つのレベルだけ扱えばよい。

➡ 配列の番号付けは不要。

新しい手法(データ分散)

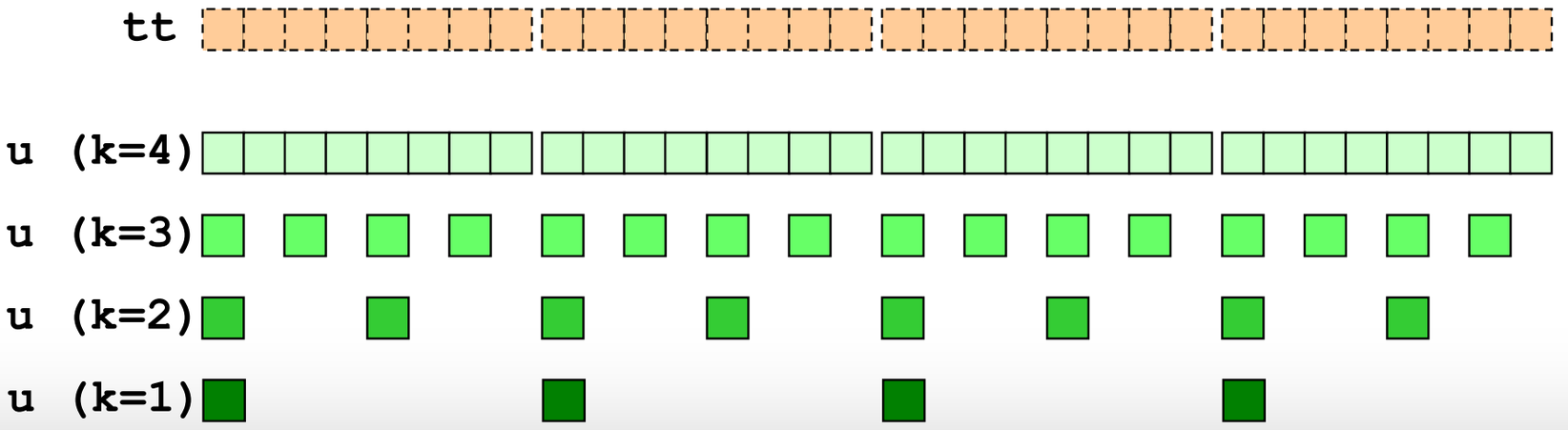
```
      recursive subroutine mg3P(u,r,k)

!HPF$ template t(N)
!HPF$ distribute (block) :: t

      double precision u(:,:,:), r(:,:,:)
!HPF$ align (*,*,i) with *t(i*(2**(LT-k))-m) :: u, r

      double precision, allocatable :: u2(:,:,:), r2(:,:,:)
!HPF$ align (*,*,i) with t(i*(2**(LT-k+1))-m2) :: u2, r2
```

- **mg3p**は、レベル*k*の配列を引数として受け取り、レベル*k-1*の配列を割り付ける。
- レベル*k*の配列は、ストライド2^{LT-k}で整列する。



整列関係を正しく指定

BLOCK分散の問題点

要素数がプロセッサ数で割り切れない場合…

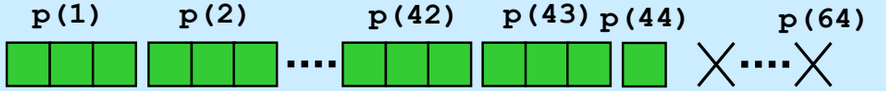
130÷64=2.03…

```
      real a(130)
!HPF$ processors p(64)
!HPF$ distribute a(block) onto p
```

GEN_BLOCKを使わなければならない

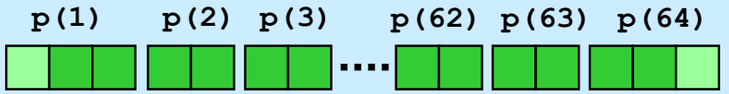
実際の分散

負荷の不均衡



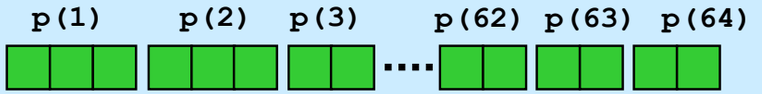

```
p(1) - p(43) : 3要素
p(44)        : 1要素
p(45) - p(64): 0要素
```

期待する分散(1)　周期境界条件を用いる場合




```
p(1)          : 3要素
p(2) - p(63): 2要素
p(64)         : 3要素
```

期待する分散(2)




```
p(1) - p(2) : 3要素
p(3) - p(64): 2要素
```

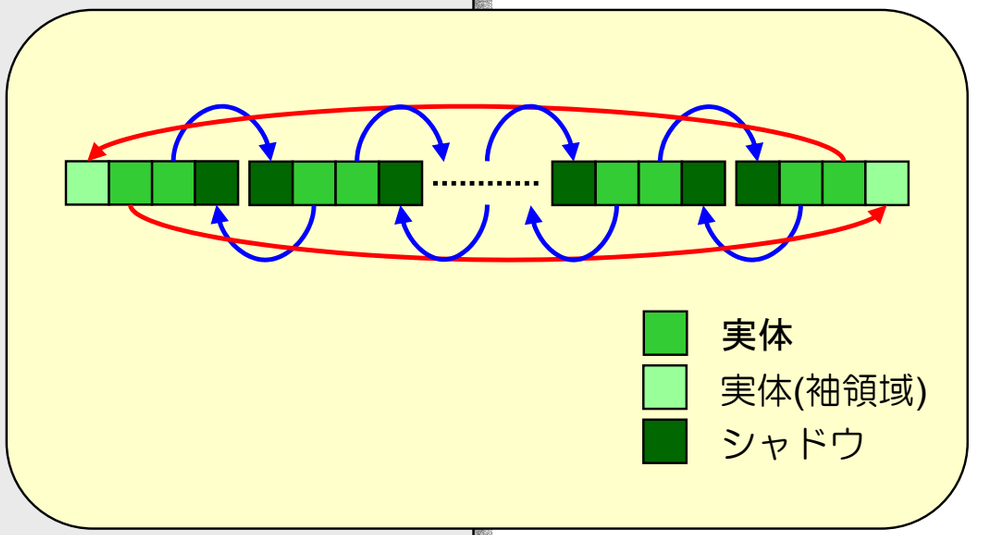
周期境界条件とシャドウ

```
      real a(130), a(130)
!HPF$ processors p(64)
!HPF$ distribute (gen_block(M)) onto p :: a, b
!HPF$ shadow (1:1) :: b

      b(1) = b(129)
      b(130) = b(2)

!HPFJ reflect b

!HPF$ independent
      do i=2, 129
!HPF$   on home(a(i)), local
        a(i) = b(i-1) + b(i) + b(i+1)
      end do
```



境界部の袖領域を更新する通信と、シャドウを更新する通信を同時に実行できないか？

→ *cyclic shift*

```
!HPF$ distribute (block) :: b
!HPF$ shadow (1:1) :: b
!HPF$ periodic (1:1) :: b
```

評価(1) 環境

- コンパイラとライブラリ
  - **HPF/ES**
    Rev.1.9.4(776) 2003/02/21
  - **FORTRAN90/ES Version 2.0**
    Rev.269 ES 13 2003/05/14
  - **MPI/ES: command Version 7.0.0 (13, February 2003)**
  - **MPI/ES: daemon Version 7.0.0 (18, February 2003)**
- 地球シミュレータのL系バッチジョブとして実行
- フラット並列化(HPFプロセッサをAPに割り当てる)

評価(2) コンパイルオプション

性能に影響するもののみ挙げる。

- **-C hopt**
- **-Wf"-pvctl vwork=stack"**
- **-Mnomapnew**
- **-Mscalarnew**
- **-Mnoerrline**
- **-Mnoentry**
- **-pi exp=..., expin=...**

評価(3) 対象コード

NPBのBT, SP, LU, CG, MG(クラスC)

- HPF版

  最新のシリアル版(NPB2.3-serial)を並列化およびチューニングしたコード

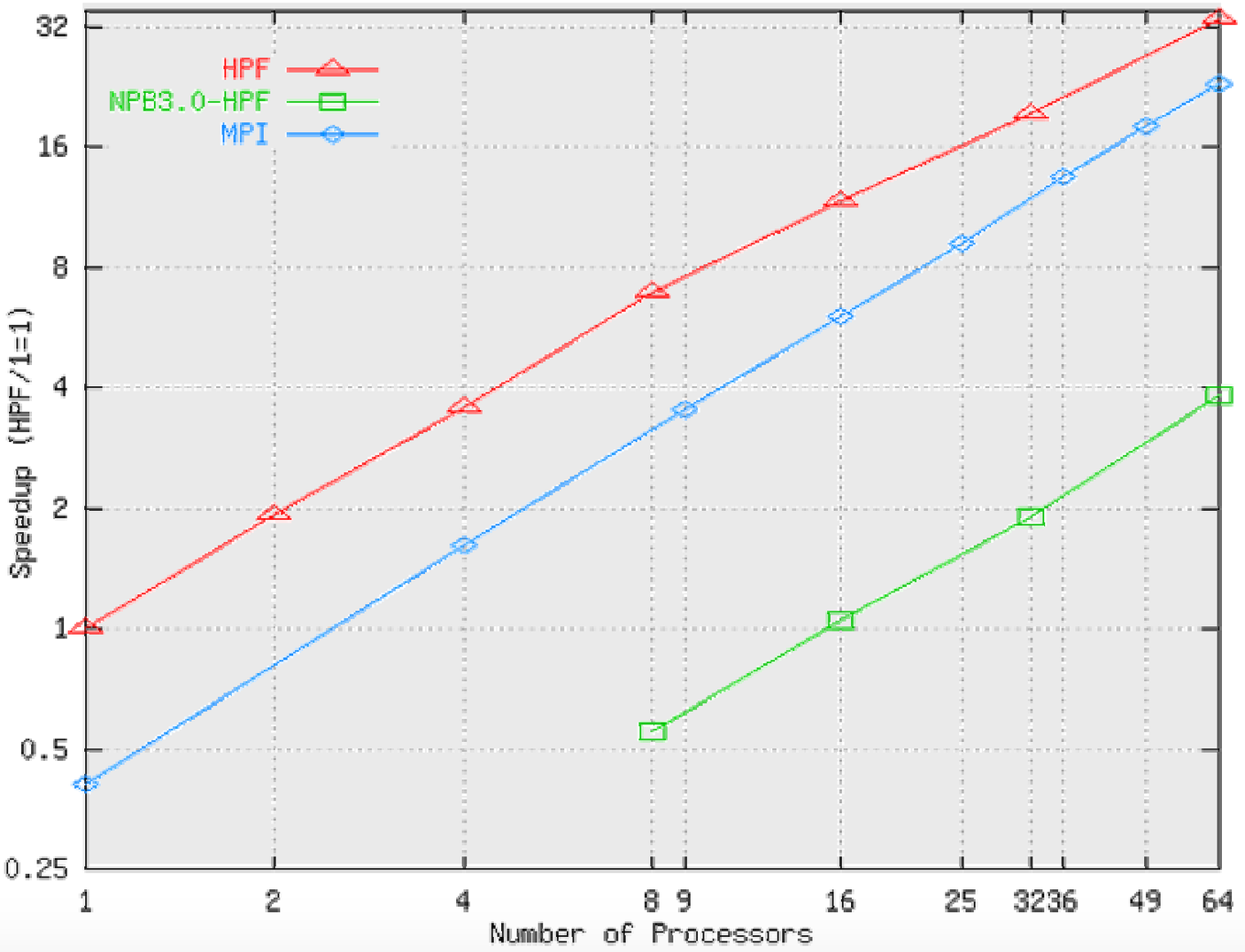
- MPI版

  NPB2.4のコード(ASIS)

- NPB3.0-HPF

  NPB3.0alphaに、必要最小限のバグ修正だけを施したコード

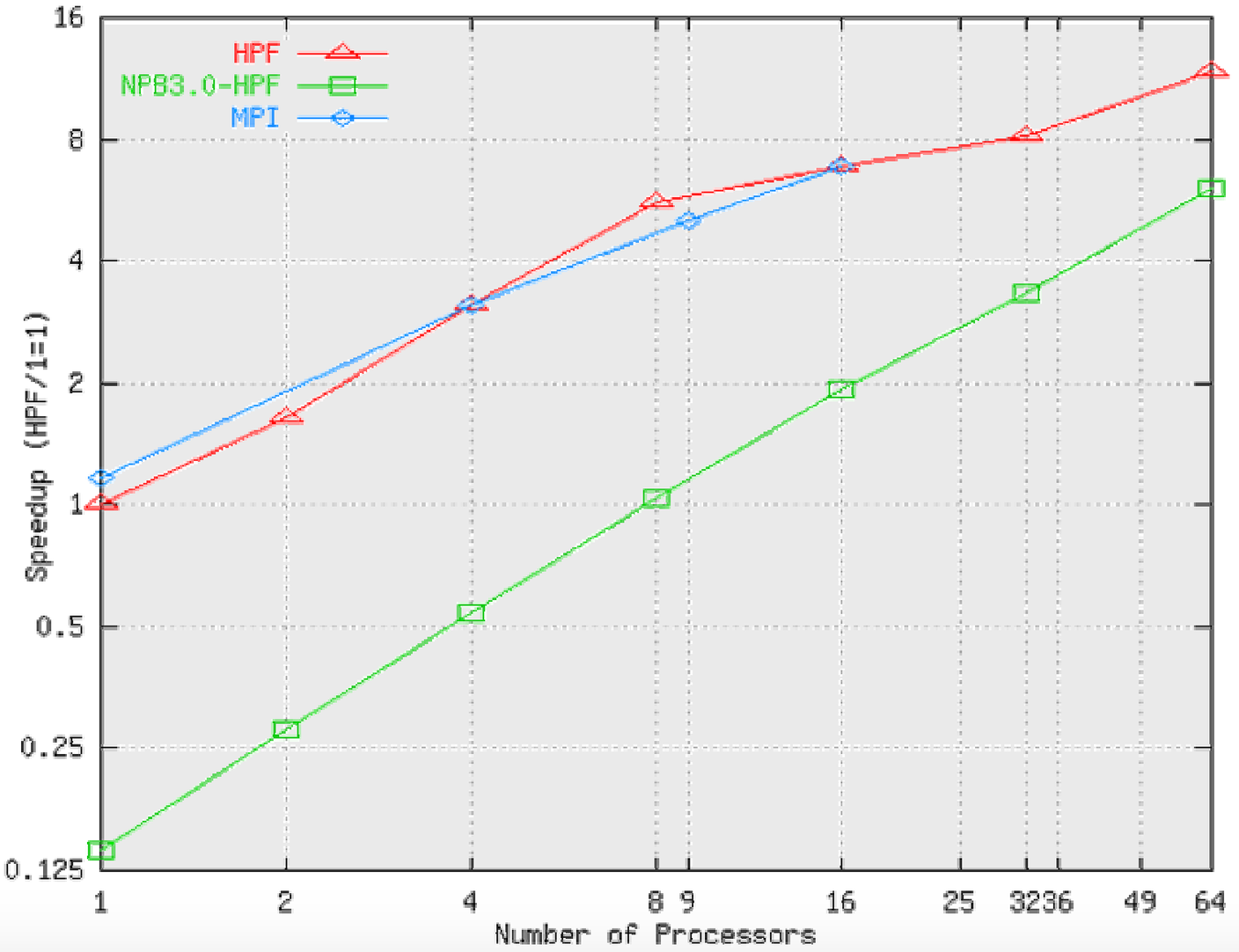
評価(4) BTベンチマーク



- スケーラビリティは比較的良好。
- MPI版との差はベクトル化率による。
- 高並列時の効率低下は、負荷の不均衡のため。

NPB3.0版の1～4CPU実行は、時間がかかりすぎるため省略。

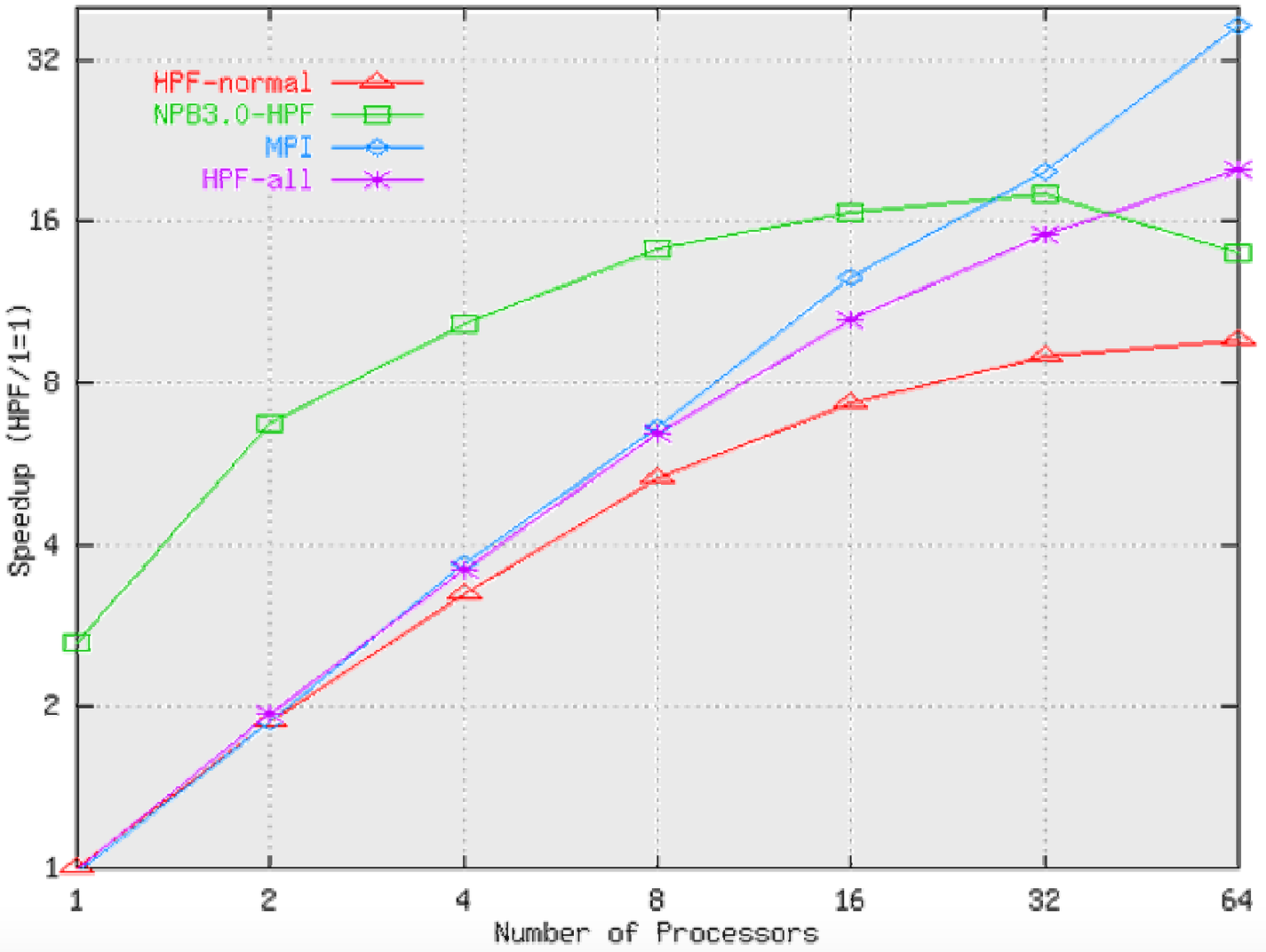
評価(5) SPベンチマーク



- MPI版とほぼ同等の性能
- NPB3.0版との差はベクトル化率による。
- 高並列時の効率低下は、負荷の不均衡のため。

MPI版の25~64CPU実行では、例外が発生する。

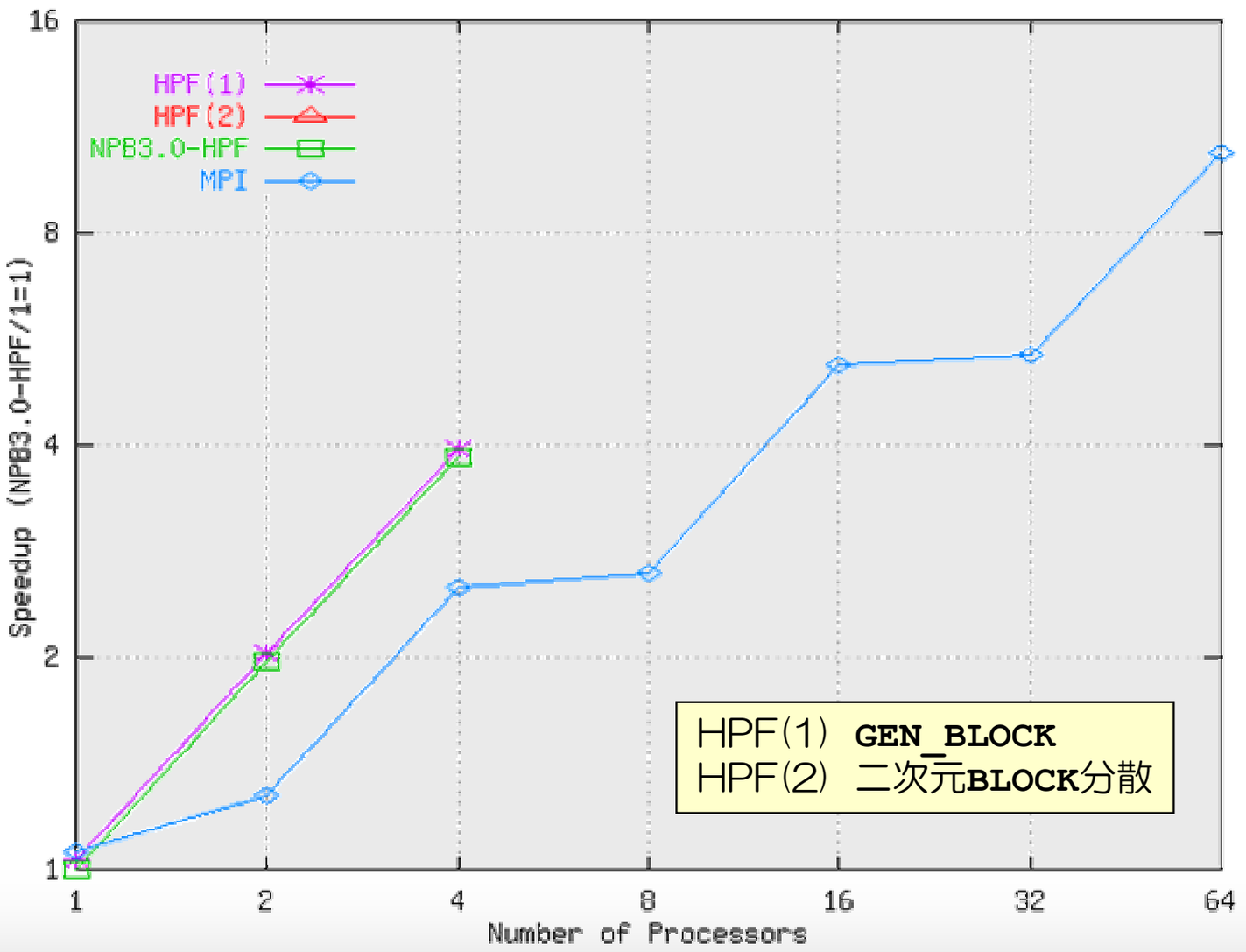
評価(6) LUベンチマーク



- 高並列時には、擬似パイプライン実行の同期処理のオーバヘッドが大きくなる。
- NPB3.0版は並列化の方法が異なる。

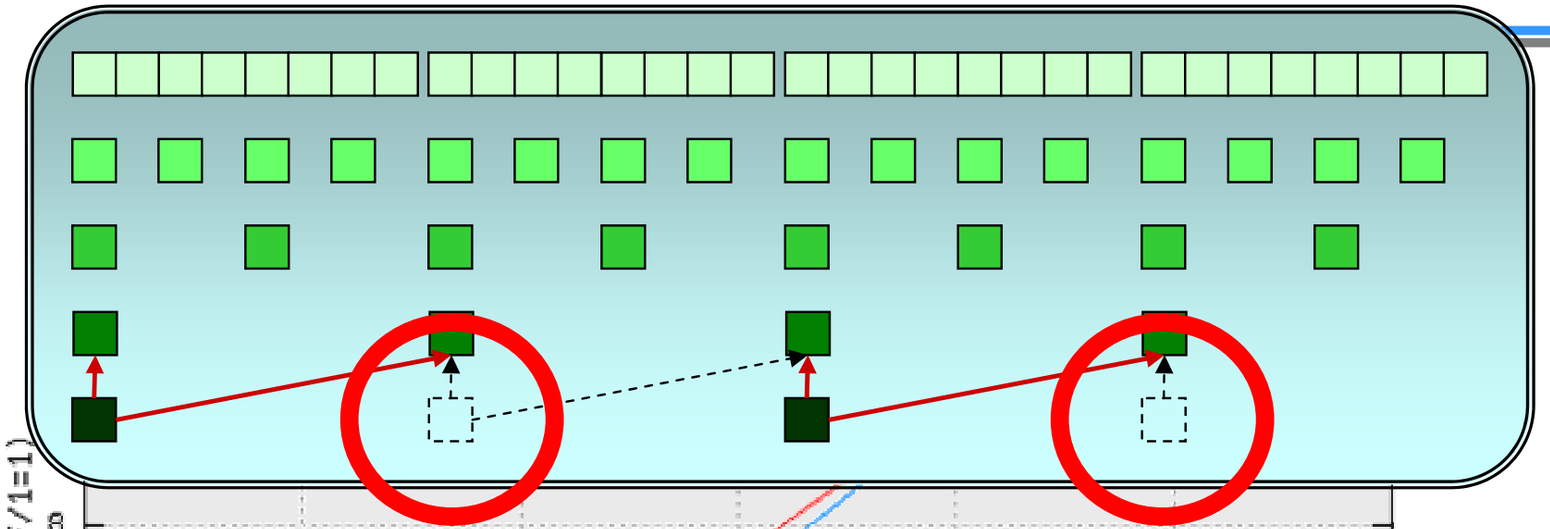
※ HPF-allでは、中間コードを手で修正して**EXT_HOME**と部分的REFLECTを実現した。

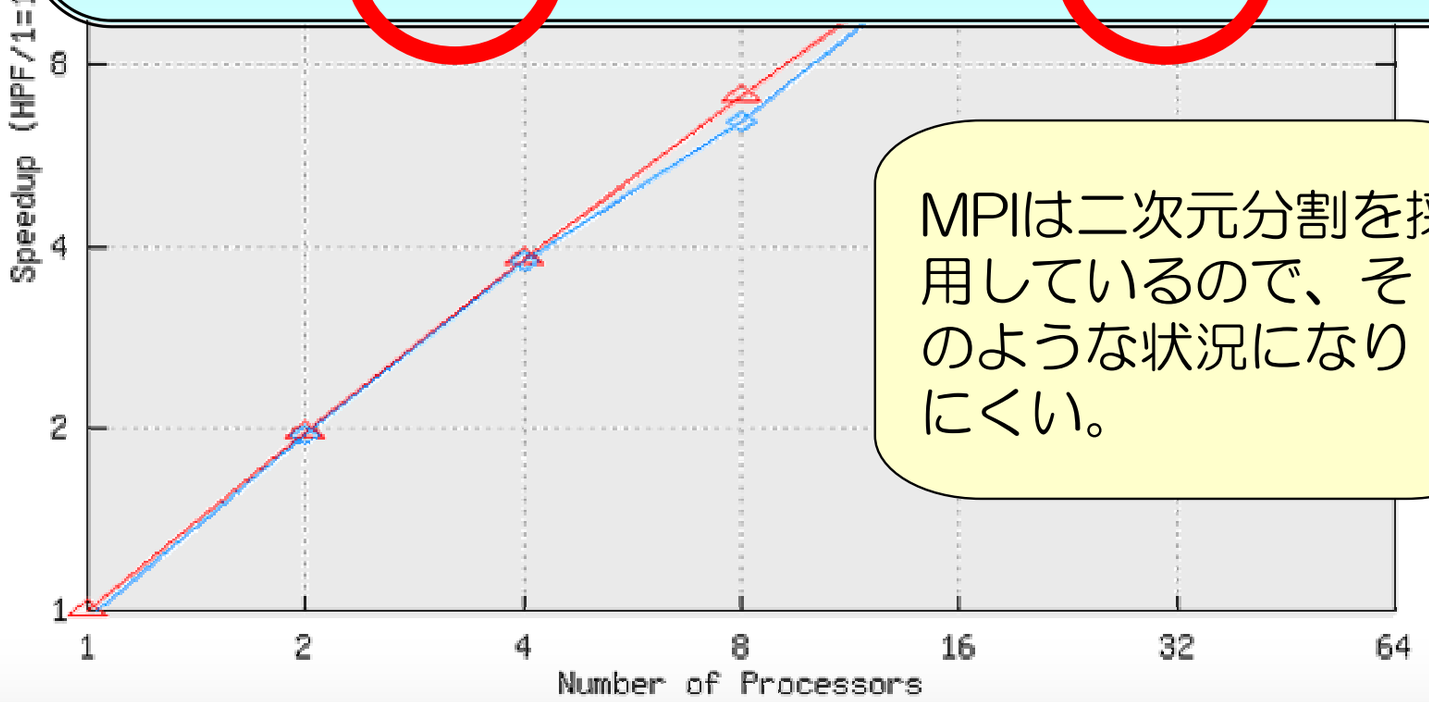
評価(7) CGベンチマーク



- HPF(2)は結果不正のため評価できず。
- MPI版が階段状になるのは、二次元分散を使っているため？
- HPF(1)とNPB3.0版は、8CPU以上でメモリ不足(疎行列の初期化)のため実行できず。

評価(8) MGベンチマーク





MPIは二次元分割を採用しているので、そのような状況になりにくい。

MPI版と同等の性能。

高並列時に効率がやや低下するのは、グリッドが疎になったときの例外的な処理による。

NPB3.0版はメモリ不足のため実行できず。

まとめ(1)

- HPF/ESによって、NPBを実装した。
  - MPI版と同じ並列化
  - ベクトル化のためのチューニング
- ESの1～8ノードで評価を行った。
  - BT, SP, MGではMPI版と同等以上、LUではほぼ同等の性能を達成した。
  - CGは結果不正のため測定できなかった。

まとめ(2) — 課題 —

HPF/ESの機能の不足や性能上の問題により、効率の低下が見られた。

- EXT_HOME
- 部分的REFLECT
- GEN_BLOCKと二次元分割の性能向上
- マルチパーティショニング
- DOACROSS型ループのパイプライン並列化
- BLOCK分散のセマンティクス(周期境界条件)