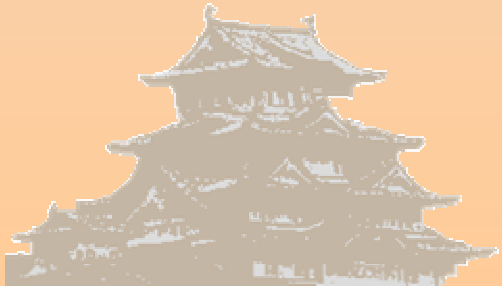
HPFによる
実用コードの並列化

姫路工業大学　坂上仁志

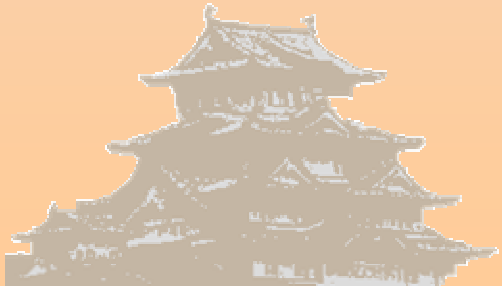
プラズマシミュレーション

- 3次元流体コード
  - レーザー核融合の爆縮過程における流体力学的不安定性の研究に使っている.
    - 球対称ターゲット
    - スタグネーション相 (減速相)
    - レイリー・テイラー不安定性
- 2次元静電粒子コード
  - プラズマの基礎的性質の研究に使っている.
    - イオン・ダブルレイヤー



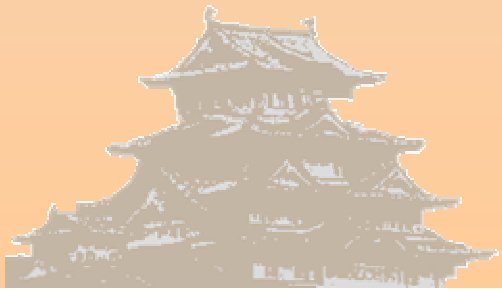
3次元流体コード

- 3次元流体方程式 (非粘性，圧縮性)
- カーテシアン座標系
- 5点差分による空間微分
- 陽的解法による時間積分
- 多次元の時間発展は分ステップ法
- 通常の領域分割で並列化可能



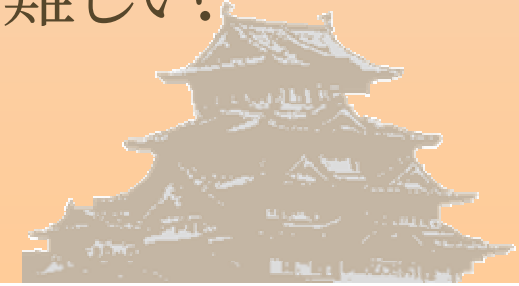
並列化の方針

- Z方向にのみ配列を分散する.
  - 1次元抽象プロセッサ
  - ベクトル処理の効率を考慮
  - Z方向の計算時のみ通信が必要
  - 多方向分散に比べて多い通信データ量
- 時間ステップの計算には，スカラ変数の全空間における値が必要である.
  - REDUCTION演算



並列化の方法

- 並列化を段階的に四つのレベルで行う.
  - DISTレベル（PROCESSORS+DISTRIBUTE）
  - INDレベル（INDEPENDENT）
  - SHADレベル（SHADOW+REFLECT）
  - LOCレベル（ON HOME LOCAL+ENDON）
- HPFでは，指示文を徐々に追加しながら，より高度な並列化ができる！
  - MPIでは，このような段階的な並列化は難しい.



DISTレベルの並列化

- PROCESSORS指示文による1次元プロセッサの定義とDISTRIBUTE (*,*,BLOCK)指示文によるデータ分散を指示する.

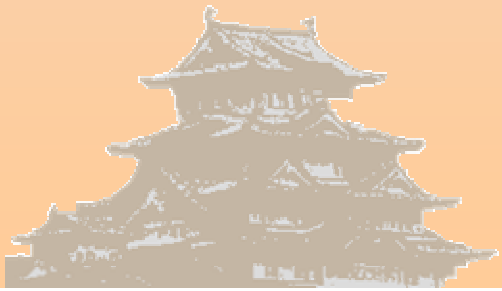
```
      parameter(lx=128,ly=128,lz=128,lpara=8)
      real sr(lx,ly,lz)
!HPF$ PROCESSORS proc(lpara)
!HPF$ DISTRIBUTE (*,*,BLOCK) ONTO proc :: sr
```

- 実際には，includeファイルに記述する.
  - 編集作業をしたHPF指示文の行数は少ない.

INDレベルの並列化

- INDEPENDENT指示文を並列実行できるDOループに追加し，並列化を促進する.

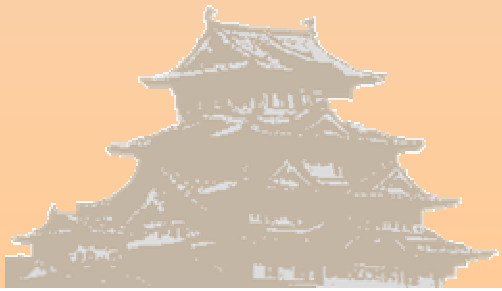
```
!HPF$ INDEPENDENT
      do iz = 1, lz-1
         do 10 iy = 1, ly
            do 10 ix = 1, lx
               wr(ix,iy,iz) = sr(ix,iy,iz) + sr(ix,iy,iz+1)
               ...
   10    continue
      end do
      ...
!HPF$ INDEPENDENT, REDUCTION(max:wram)
      do iz = 1, lz
         do 20 iy = 1, ly
            do 20 ix = 1, lx
               ...
               wram = max(wram, ... , ... )
   20    continue
      end do
```



SHADレベルの並列化

- SHADOW+REFLECT指示文により通信を最適化する.
  - ブロック通信の促進と重複通信の抑止

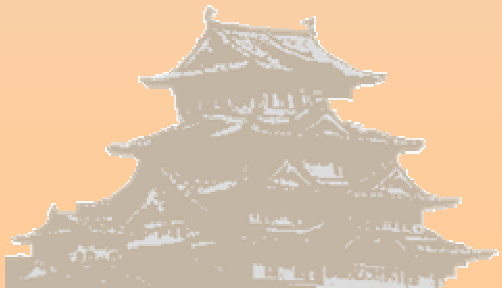
```
!HPF$ SHADOW (0:0,0:0,0:1) :: sr
!HPF$ SHADOW (0:0,0:0,1:0) :: sp
!HPF$ SHADOW (0:0,0:0,1:1) :: se
      ...
      ...
!HPFJ REFLECT sr
!HPFJ REFLECT sp
!HPFJ REFLECT se
!HPF$ INDEPENDENT
      do iz = 2,lz-1
         do 30 iy = 1,ly
            do 30 ix = 1,lx
               wr(ix,iy,iz) = sr(ix,iy,iz) + sr(ix,iy,iz+1)
               wp(ix,iy,iz) = sp(ix,iy,iz) + sp(ix,iy,iz-1)
               we(ix,iy,iz) = se(ix,iy,iz-1) + se(ix,iy,iz+1)
               ...
   30    continue
      end do
C**** REFLECT sr
!HPF$ INDEPENDENT
      do iz = 1,lz-1
         do 40 iy = 1,ly
            do 40 ix = 1,lx
               ... = sr(ix,iy,iz) + sr(ix,iy,iz+1)
               ...
   40    continue
      end do
```



LOCレベルの並列化

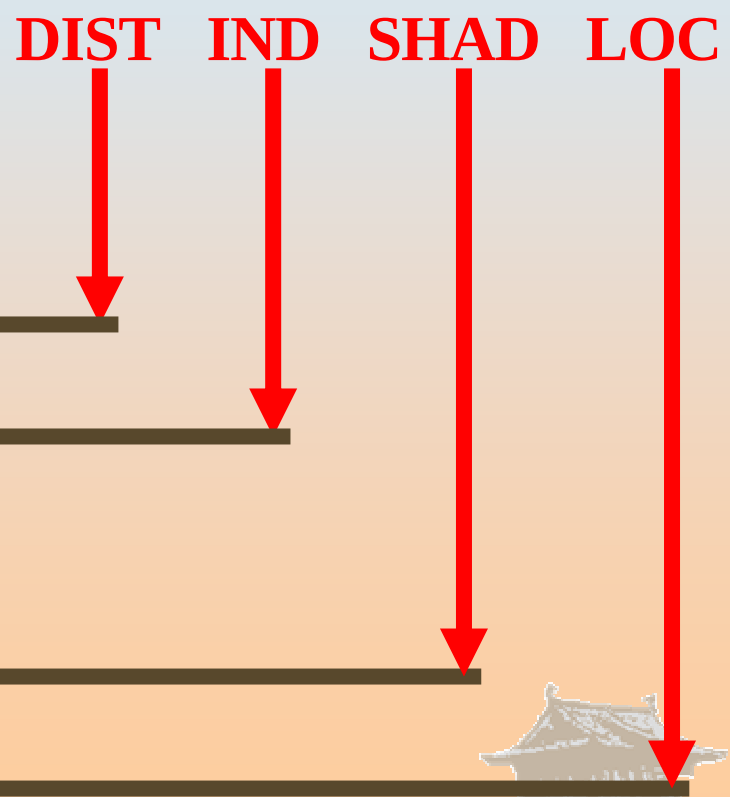
- LOCAL指示文により不必要な通信を明示的に抑制する.

```
!HPFJ REFLECT sr
!HPF$ INDEPENDENT
      do iz = 1, lz-1
!HPFJ ON HOME(wr(:,:,iz)), LOCAL BEGIN
         do 10 iy = 1, ly
            do 10 ix = 1, lx
               wr(ix,iy,iz) = sr(ix,iy,iz) + sr(ix,iy,iz+1)
               ...
   10    continue
!HPFJ end on
      end do
```



挿入したHPF指示文の行数

オリジナル	1244
PROCESSORS	7
DISTRIBUTE	31
INDEPENDENT	23
SHADOW	23
REFLECT	6
ON-LOCAL/ENDON	44



* includeファイルを展開して計数

NEC SX-4の実行時間［秒］

- Lx=Ly=Lz=128

proc#	DIST	IND	SHAD	LOC
1	82.69	82.71	83.16	81.40
2	41.50	41.53	41.85	40.92
4	20.93	20.94	21.23	20.64
8	10.59	10.58	10.87	10.47
16	5.45	5.45	5.74	5.54
# 32	2.85	2.89	3.25	2.92
# 64	2.01	1.71	2.39	2.06
! # 32	18.59	18.19	19.17	18.61
! # 64	11.11	9.80	11.65	10.29

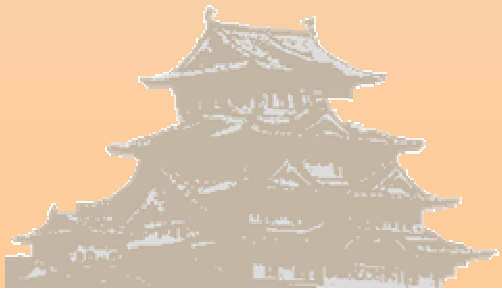
#：ノード間通信，!：L=256

NEC SX-5の実行時間［秒］

- Lx=Ly=Lz=128

proc#	DIST	IND	SHAD	LOC
1	26.32	27.90	26.69	27.38
2	13.25	13.79	13.55	13.30
4	6.76	6.81	6.90	6.78
8	3.52	3.49	3.58	3.50
16	1.92	1.91	2.04	1.95

＊すべてノード内通信



富士通 VPP800の実行時間［秒］

- Lx=Ly=Lz=128
- DISTレベル
  - ループの自動並列化が不可
- INDレベル
  - ループ内で無駄な通信が多発
- SHADレベル
  - 袖領域に対する非効率通信
- LOCレベル
  - 無駄な通信の抑止
  - 効率的な並列実行

proc#	LOC
1	31.33
2	15.85
4	8.12
8	4.13
16	2.30
32	1.58
! 16	11.82
! 32	6.47

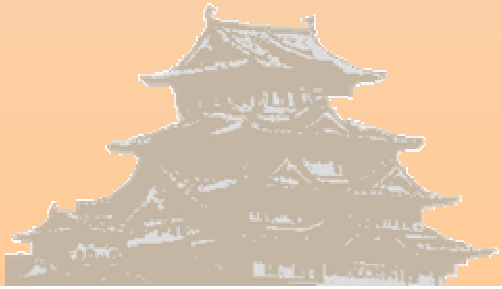
＊すべてノード間通信, ! : L=256

日立 SR8000の実行時間 秒]

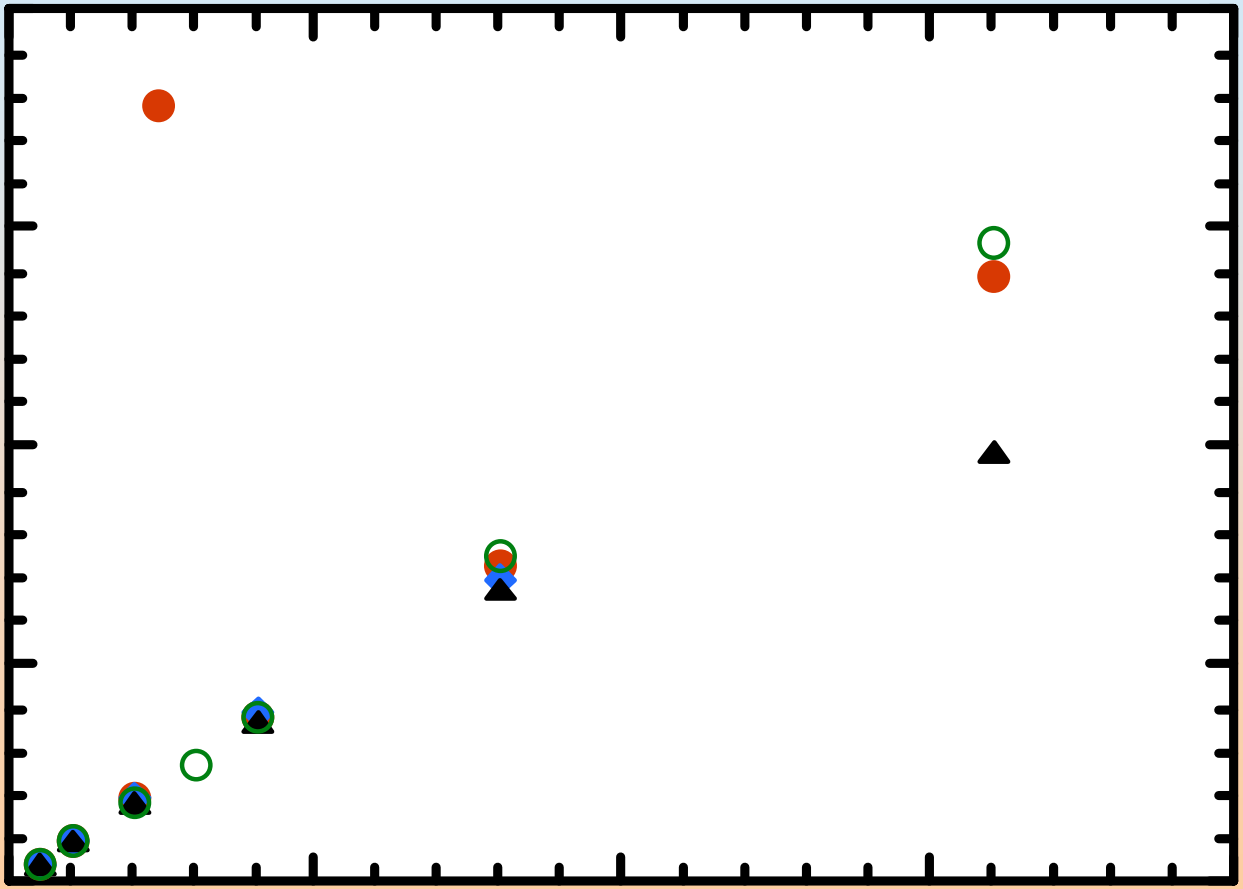
- Lx=Ly=Lz=127
  - キャッシュミスによる極度の性能低下
- LOC level
  - LOCAL指示文をサポートしていない.

proc#	DIST	IND	SHAD
1	319.59	317.76	317.83
2	160.76	160.75	160.74
4	81.98	82.21	82.11
8	41.72	41.71	41.76
# 16	21.16	21.16	21.16
# 32	10.81	10.81	10.81
# 64	5.70	5.68	5.76
# 128	3.27	3.20	3.20

: ノード間通信

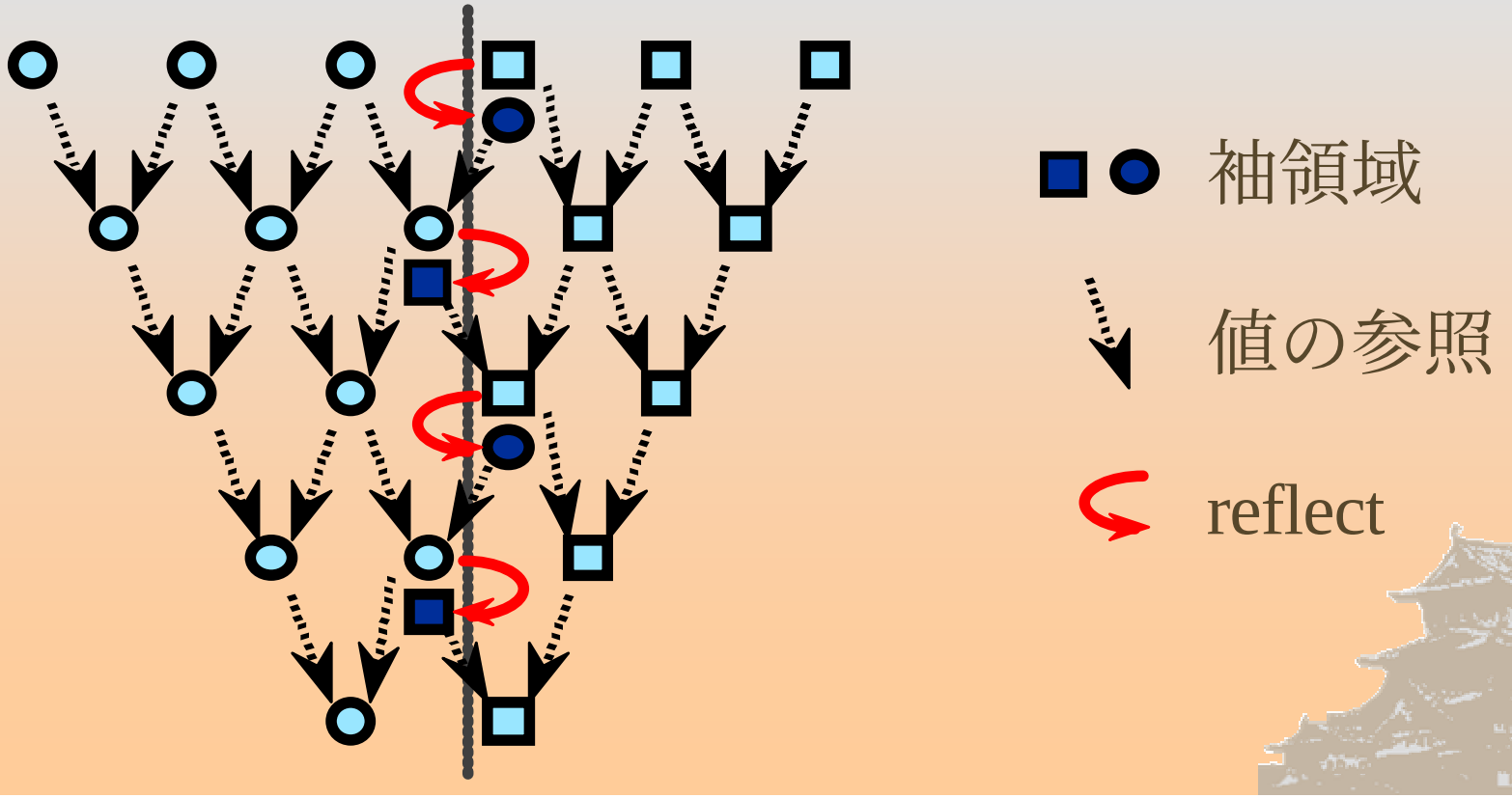


並列化によるスピードアップ



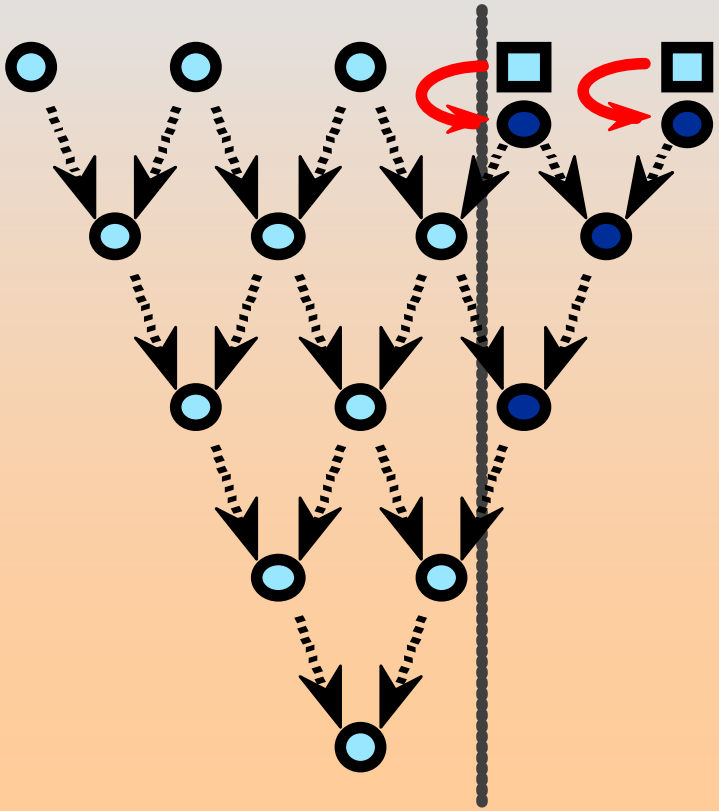
LOCレベルにおける通信

- SHADOWとREFLECT指示文により明示的に通信を指示している.
  - 境界において，REFLECTが4回必要

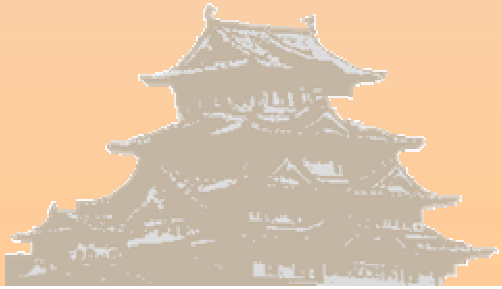


EXT_HOME指示文による通信の最適化

- プロセッサ境界を拡張することで袖領域の計算を重複して行い，REFLECTの回数を一回に減らすことができる.



通信の初期化が通信速度に比べて遅いネットワークでは有効である. ただし，重複計算のための新たなオーバーヘッドは発生する.

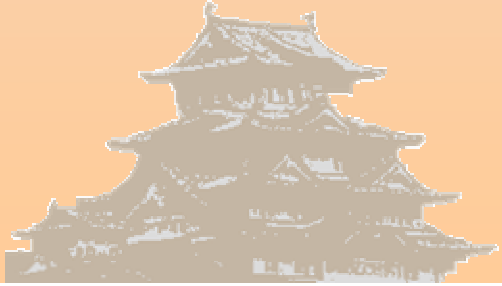


EXT_HOME指示文を用いた 富士通 VPP800の実行時間 秒］

- 富士通のみがサポートしている.
- 若干の性能向上が見られるが(proc# ≤ 5), 重複計算のオーバーヘッドが通信の最適化による効果より大きい(proc# ≥ 10).

proc#	LOC	EXT
1	52.524	52.734
2	26.367	26.353
5	10.675	10.649
10	5.370	5.418
20	2.837	2.886
40	1.517	1.580

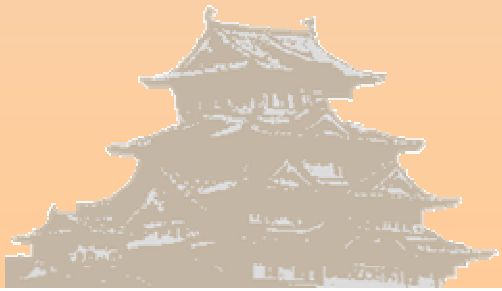
Lx=Ly=Lz=280



非同期通信

- 非同期通信を使うと，データの転送と計算を同時に実行できる.
  - DOループ実行の前に6変数のREFLECTが必要である.
  - DOループを二つに分割し，非同期転送を使う.

```
!HPFJ REFLECT sr, sm
!HPFJ ASYNC(id) BEGIN
!HPFJ REFLECT sn, sl, se, sp
!HPFJ END ASYNC
!HPF$ INDEPENDENT
      do iz = 1,lz-1
!HPFJ ON EXT_HOME( wtmp(:,:,iz)), LOCAL BEGIN
        (sr, smだけを使った計算)
!HPFJ END ON
      end do
!HPFJ ASYNCWAIT(id)
!HPF$ INDEPENDENT
      do iz = 1,lz-1
!HPFJ ON EXT_HOME( wtmp(:,:,iz)), LOCAL BEGIN
        (すべての変数を使った計算)
!HPFJ END ON
      end do
```

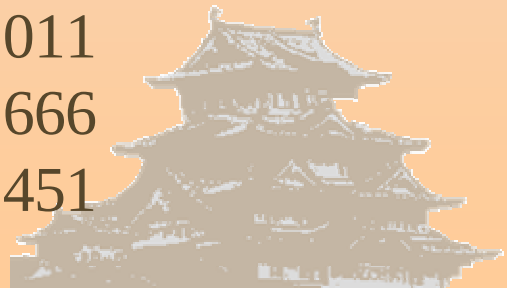


非同期通信を用いた
富士通 VPP800の実行時間 秒］

- 富士通のみがサポートしている.
- 富士通はハードウェアによりSHIFT型通信を効率よく行うため，性能に顕著な差はない.

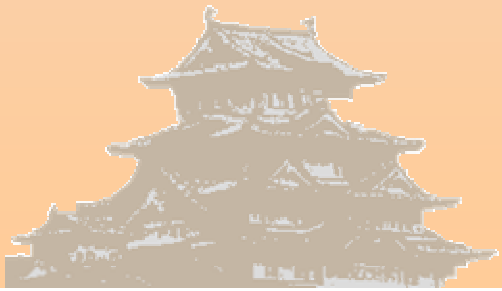
proc#	EXT	REF2	REF3	REF4	REF5
1	49.145	49.079	48.859	48.399	48.505
2	24.584	24.512	24.422	24.188	24.259
5	9.929	9.913	9.868	9.775	9.806
10	5.080	5.056	5.044	4.999	5.011
20	2.703	2.684	2.676	2.652	2.666
40	1.483	1.468	1.459	1.444	1.451

Lx=Ly=265,
Lz=280



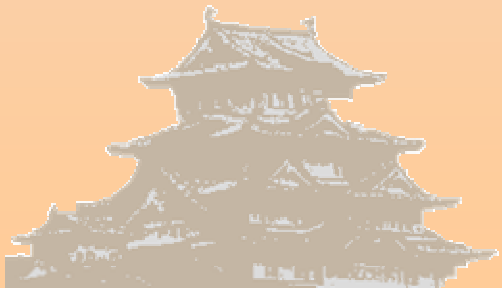
地球シミュレータにおけるHPF

- DISTRIBUTE指示文だけを挿入し，あとはコンパイラの自動並列化に任せる．
  - INDEPENDENT指示文の記述は必要ない．
  - ただし，集約演算のDOループのみ必要である．
- 自動SHADOWを抑制し，袖領域を最適化する．
  - 自動SHADOWでは，両側に4つの袖を確保している．
  - SHADOW指示文により袖領域を明示的に確保する．
- SHIFT型通信は，コンパイラの最適化に任せる．
  - REFLECT指示文は書かない．



自動並列化におけるHPF指示文

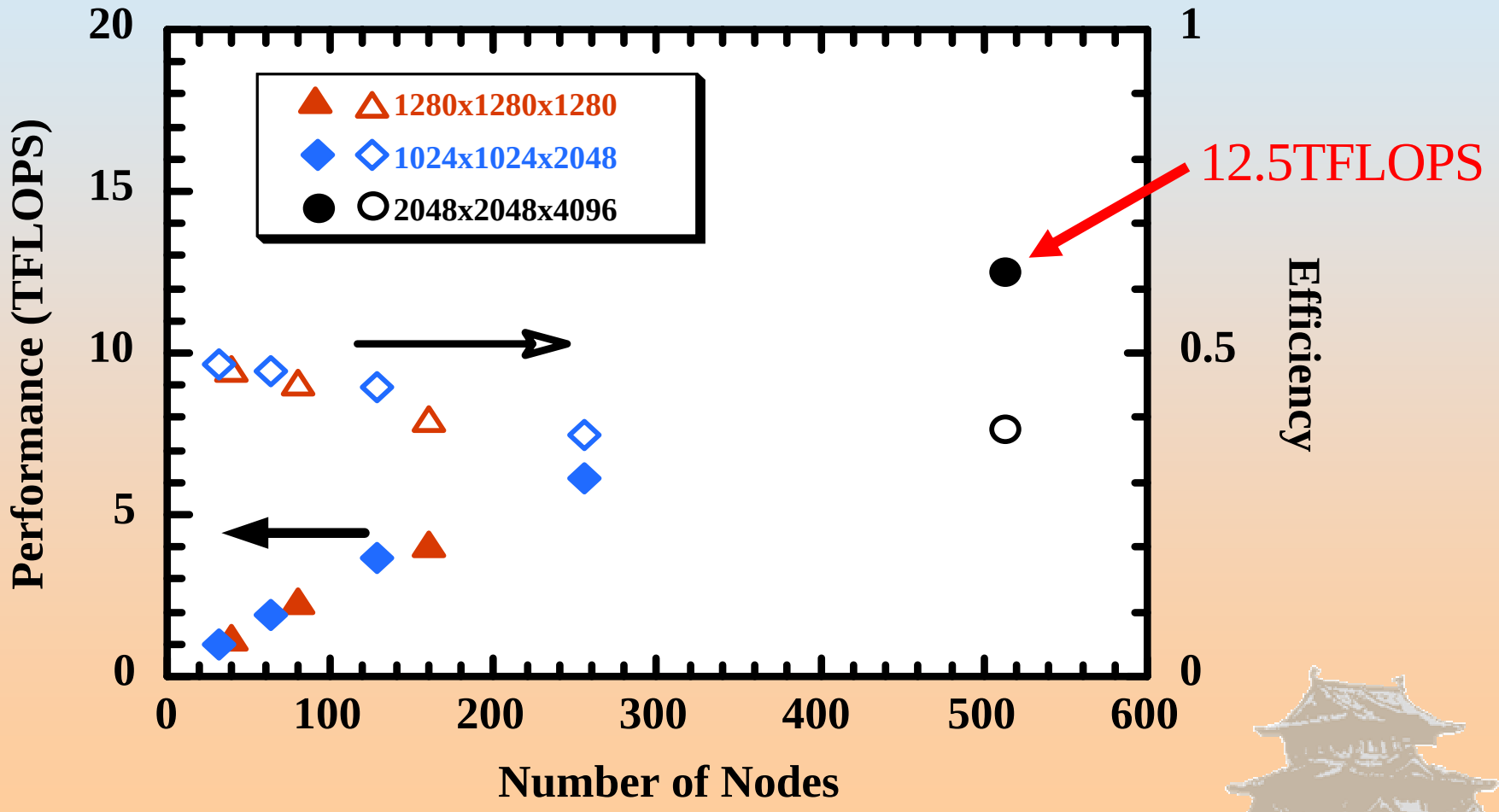
```
      parameter(lx=1024, ly=1024, lz=1024)
!HPF$ PROCESSORS es(NUMBER_OF_PROCESSORS())
      dimension sr(lx,ly,lz), sm(lx,ly,lz), sv(lx,ly,lz)
!HPF$ DISTRIBUTE (*,*,BLOCK) onto es :: sr, sm, sv
!HPF$ SHADOW (0,0,0:1) :: sr, sm
      …
      do iz = 1, lz-1
        do iy = 1, ly
          do ix = 1, lx
            …
            sv(ix,iy,iz) = sm(ix,iy,iz+1) / sr(ix,iy,iz+1)
            …
           end do
        end do
      end do
c-----
!HPF$ INDEPENDENT, REDUCTION(MAX:sram)
      do iz = 1, lz
        do iy = 1, ly
          do ix = 1, lx
            …
            sram = max( sram, … )
          end do
        end do
      end do
```



自動並列化における通信

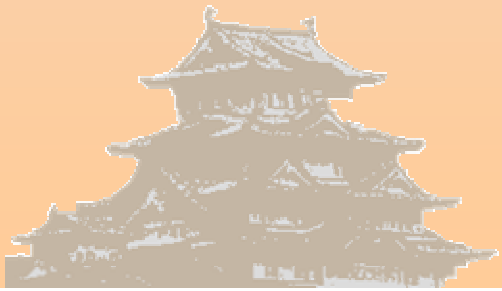
- HPF/ESコンパイラは，以下の手法により，自動的に効率の良いSHIFT型通信を実現している．
  - 不必要なデータ転送を削除する．
  - 異なった変数に対する複数のデータ転送を可能なかぎり一つにまとめる．
  - 同一形状および分散を持つ異なった配列変数間で，通信スケジュールを共有する．
  - サブルーチン内では，通信スケジュールを再利用する．

自動並列化における性能



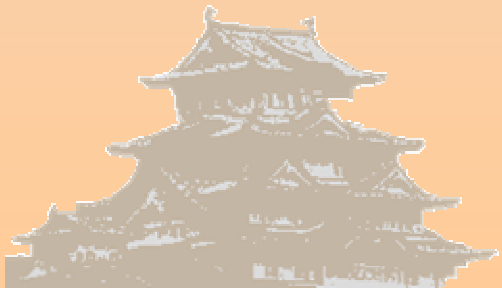
手動最適化

- 並列実行時に発生するDOループ内の不必要な通信を抑制する.
  - ON HOME指示文のLOCAL節
- 袖部分の通信をユーザが明示的に指示して,不必要な通信を削除する.
  - REFLECT指示文



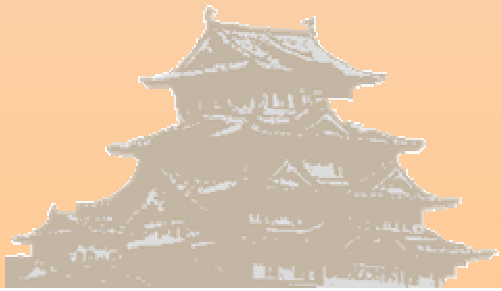
手動最適化におけるHPF指示文

```
      …
!HPF$ REFLECT sm, sr
      do iz = 1, lz-1
!HPF$ ON HOME(sv(:,:,iz)), LOCAL BEGIN
        do iy = 1, ly
          do ix = 1, lx
            …
            sv(ix,iy,iz) = sm(ix,iy,iz+1) / sr(ix,iy,iz+1)
            …
          end do
        end do
!HPF$ END ON
      end do
      …
```

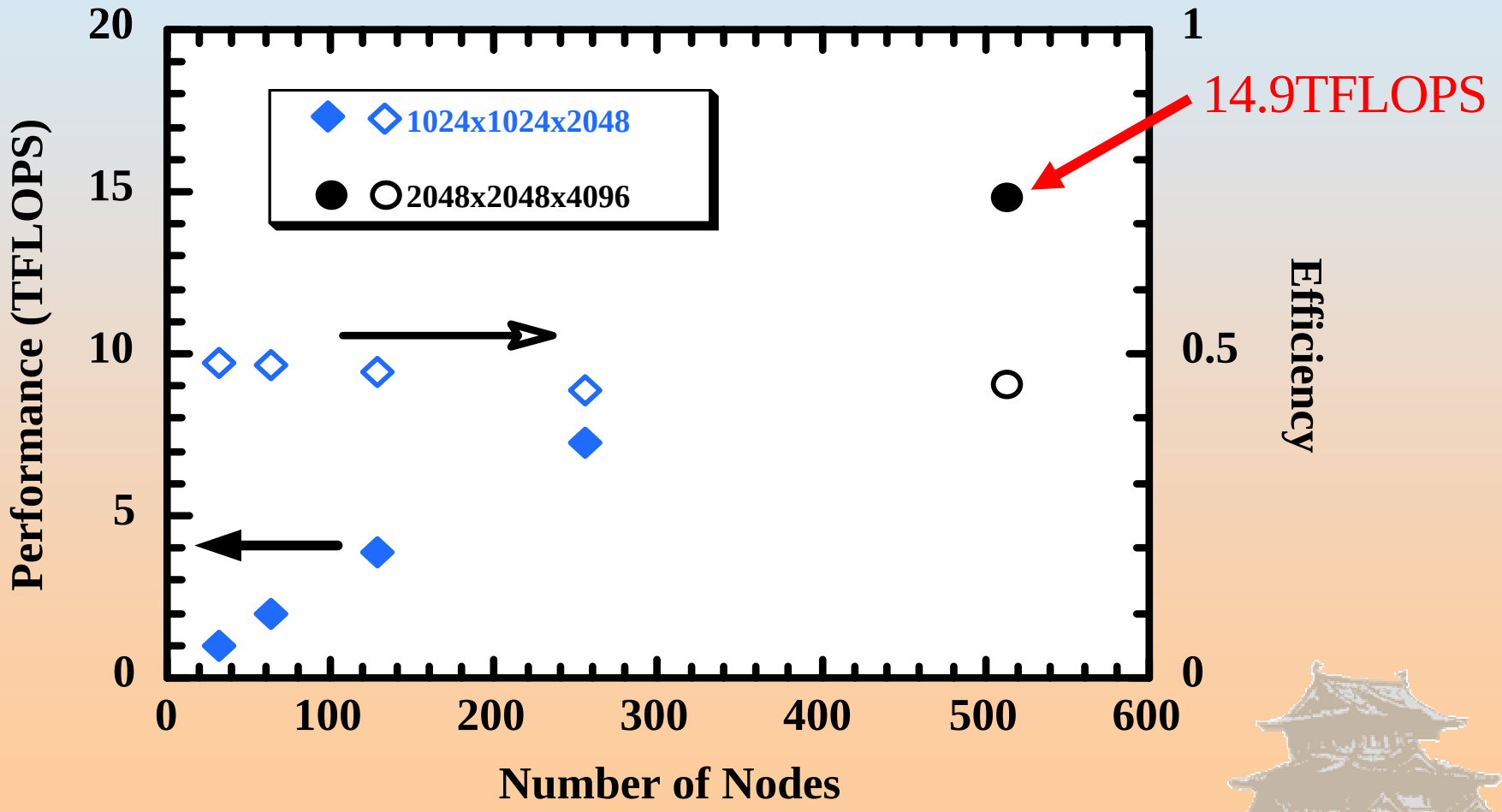


手動最適化の効果

- LOCAL, REFLECTによる最適化は，コンパイラにより自動的に行われていた.
- 自動並列化では，通信スケジュールは，サブルーチンの呼び出し毎に少なくとも一回，実行情報を元に作成される.
- 手動最適化では，コンパイラが通信スケジュールを静的に決定できるため，全体の実行を通じて一回作成するだけである.



手動最適化における性能

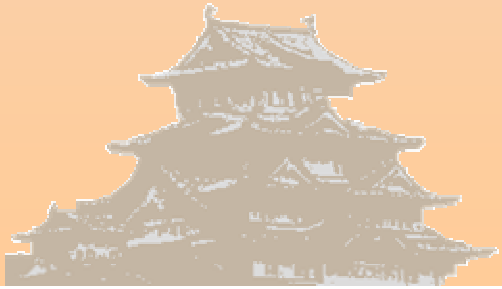


3次元流体コードのまとめ

- NEC，富士通，日立のHPFコンパイラは，高い並列性能を実現している.
- 富士通においてHPF/JAを用いた最適化を試みたが，顕著な性能向上はなかった．しかし，台数が増えると，このような最適化は重要になる.
  - アムダールの法則
- 地球シミュレータのHPFコンパイラは，超並列時でも高い並列性能を実現している.

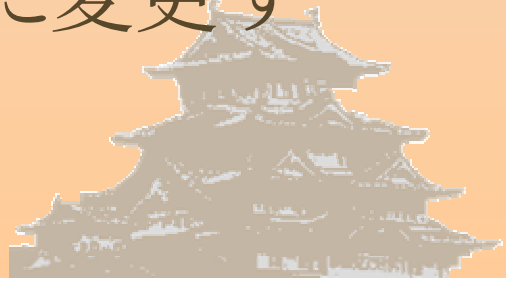
2次元静電粒子コード

- 典型的なParticle In Cell法
- 電子とイオンの運動方程式
- 電荷密度の計算
  - 間接インデックスを用いたランダムな配列アクセス
- 2次元ポアソン方程式
  - 2次元フーリエ変換／逆変換
  - 1次元FFTルーチンの多重呼び出し

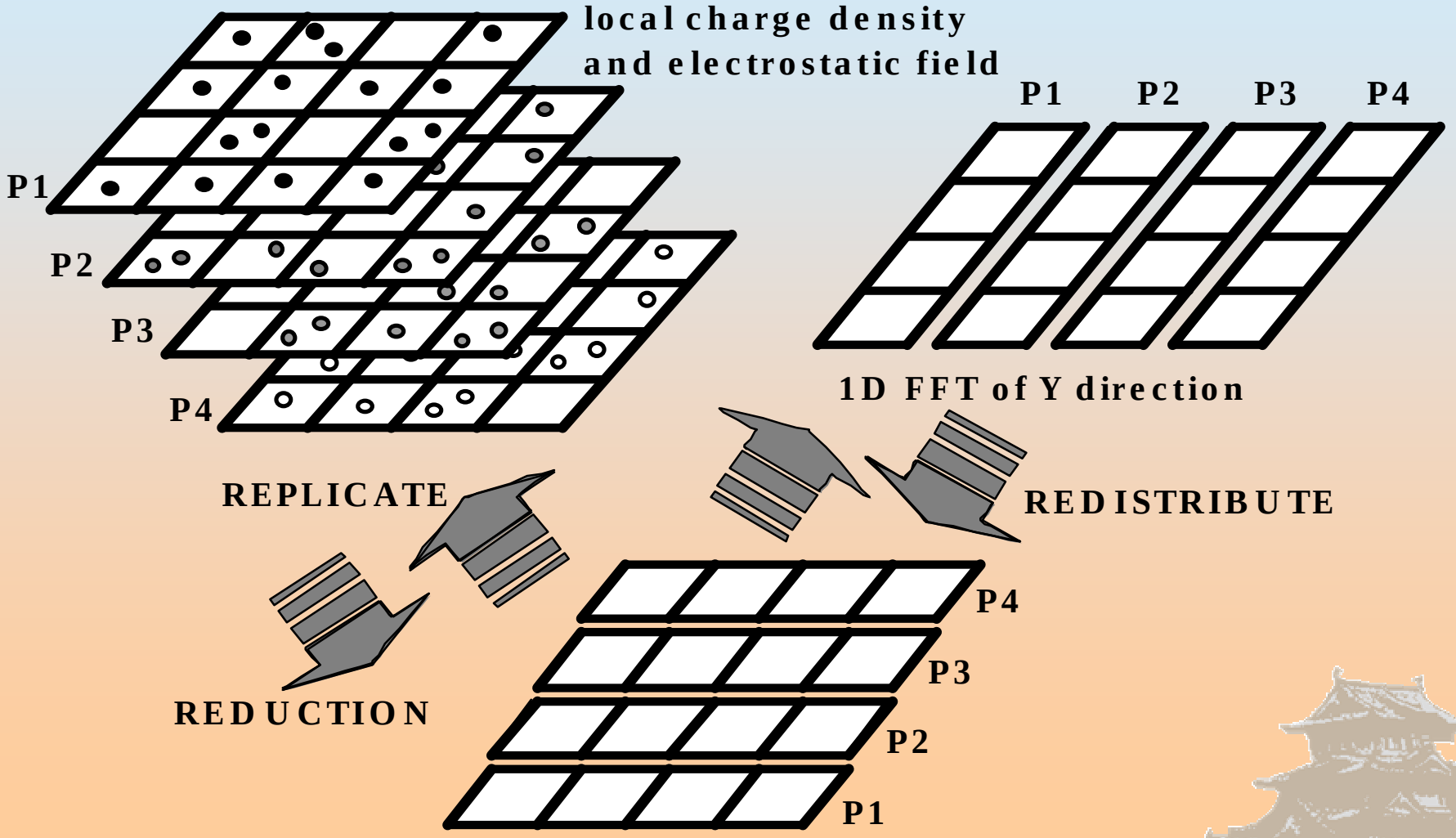


並列化の方針

- 粒子データを分散する.
  - 領域を分散するためには，トリッキーなコーディングが必要である.
- 一時配列の導入により電荷密度の並列計算を可能にする.
- 1次元FFTルーチンの並列呼び出しにより，2次元FFTを並列化する.
- 場データの分散を必要に応じて動的に変更する.



場データの動的再分散


local charge density
and electrostatic field
P1
P2
P3
P4
P1
P2
P3
P4
1D FFT of Y direction
REPLICATE
REDISTRIBUTE
REDUCTION
P4
P3
P2
P1
sum up for total charge density and 1D FFT of X direction

電荷密度計算のHPF化

ベクトル化

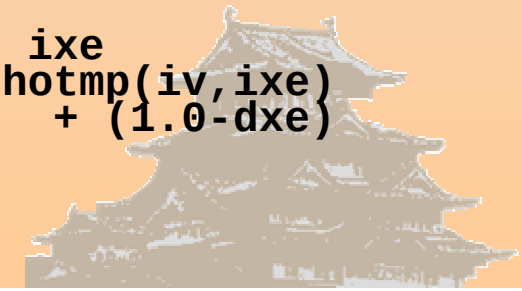
```
      real xe(no), rho(lx)
      do i = 1, no
         ixe = xe(i)
         dxe = xe(i) - ixe
         rho(ixe) = rho(ixe)
     &            + (1.0-dxe)
      end do
```

```
      real xe(no), rho(lx), rhotmp(lvec,lx)
      do iv = 1, lvec
         do i = 1, no, lvec
            ixe = xe(i+iv-1)
            dxe = xe(i+iv-1) - ixe
            rhotmp(iv,ixe) = rhotmp(iv,ixe)
     &                       + (1.0-dxe)
         end do
      end do
c.... reduction rhotmp to rho
      do iv = 1, lvec
         do i = 1, lx
            rho(i) = rho(i) + rhotmp(iv,i)
         end do
      end do
```

HPF化

```
      real xe(no), rho(lx), rhotmp(lvec,lx)
!HPF$ DISTRIBUTE xe(CYCLIC),rhotmp(BLOCK,*)
      do iv = 1, lvec
!HPF$ ON HOME(rhotmp(iv,:)), LOCAL BEGIN
         do i = 1, no, lvec
            ixe = xe(i+iv-1)
            dxe = xe(i+iv-1) - ixe
            rhotmp(iv,ixe) = rhotmp(iv,ixe)
     &                       + (1.0-dxe)
         end do
!HPF$ ENDON
      end do
```

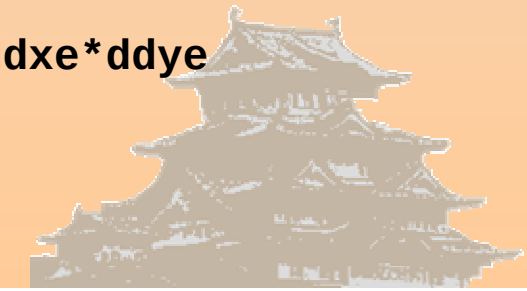
- ソースは無修正だが，並列性能が出ない！



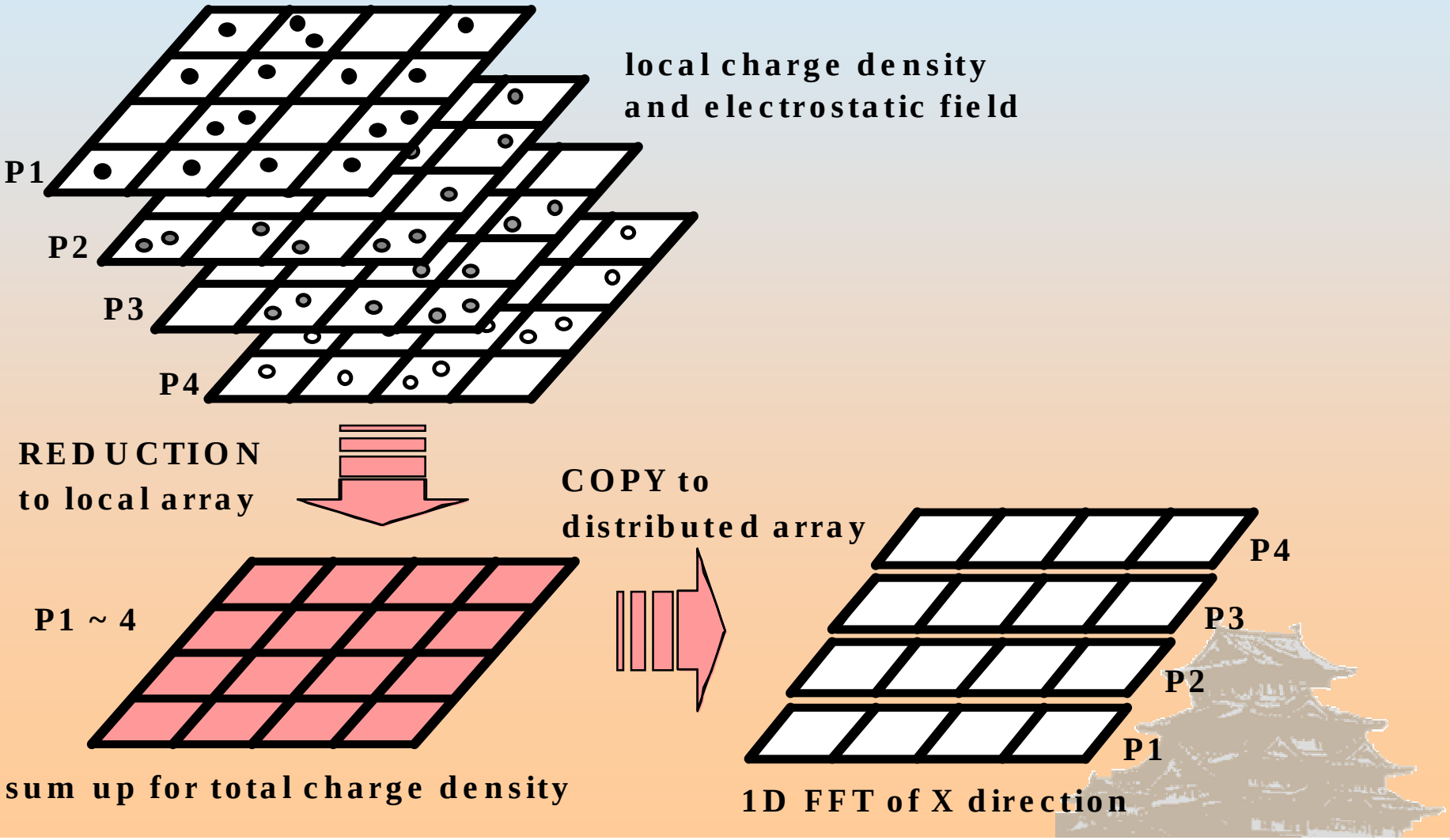
次元拡張によるHPF化

```
!HPF$ PROCESSORS proc(lpara)
      real xe(no/lpara,lpara),ye(no/lpara,lpara)
      real rhotmp(lx,ly,lpara)    ! rhotmp(lvec,lx,ly)
!HPF$ DISTRIBUTE (*,BLOCK) :: xe,ye
!HPF$ DISTRIBUTE (*,*,BLOCK) :: rhotmp
!HPF$ INDEPENDENT
      do ip = 1, lpara
!HPF$ ON HOME(xe(:,ip)),LOCAL BEGIN
         do 10 i = 1, no/lpara
            ixe  = xe(i,ip)
            dxe  = xe(i,ip) - ixe
            ddxe = 1.0-dxe
            iye  = ye(i,ip)
            dye  = ye(i,ip) - iye
            ddye = 1.0 - dye
            rhotmp(ixe,iye,ip) = rhotmp(ixe,iye,ip)-ddxe*ddye
   10 continue
!HPF$ END ON
      end do
```

- ベクトル化できない！
- 他のループでもソース修正が必要！

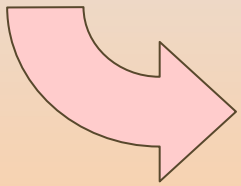


分散配列REDUCTIONの回避

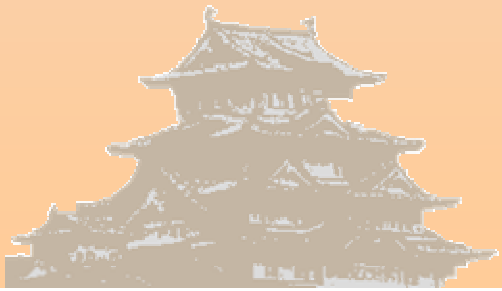


REDUCTIONのHPF指示文

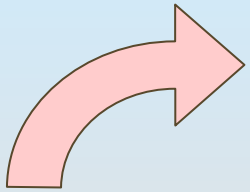
```
!HPF$ PROCESSORS proc(lpara)
!HPF$ DISTRIBUTE (*,BLOCK) ONTO proc :: rho
!HPF$ DISTRIBUTE (*,*,BLOCK) ONTO proc :: rhotmp
!HPF$ INDEPENDENT,REDUCTION(+:rho)
      do ip = 1,lpara
         do j = 1,ly
            do i = 1,lx
               rho(i,j) = rho(i,j) + rhotmp(i,j,ip)
            end do
         end do
      end do
```



```
!HPF$ PROCESSORS proc(lpara)
!HPF$ DISTRIBUTE (*,BLOCK) ONTO proc :: rho
!HPF$ DISTRIBUTE (*,*,BLOCK) ONTO proc :: rhotmp
!HPF$ PROCESSORS procdummy
!HPF$ DISTRIBUTE (*,*) ONTO procdummy :: trho
!HPF$ INDEPENDENT,REDUCTION(+:trho)
      do ip = 1,lpara
         do j = 1,ly
            do i = 1,lx
               trho(i,j) = trho(i,j) + rhotmp(i,j,ip)
            end do
         end do
      end do
!HPF$ INDEPENDENT
      do j = 1,ly
         do i = 1,lx
            rho(i,j) = trho(i,j)
         end do
      end do
```

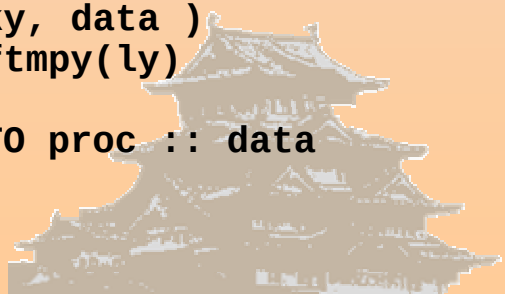


FFTルーチンのHPF指示文



```
subroutine fftfx2 ( kx, ky, data)
dimension data(kx,ky)
do iy = 1, ky
   call fftf1 ( kx, data(1,iy) )
end do
return
end
subroutine fftfy2 ( kx, ky, data)
dimension data(kx,ky), ftmpy(ly)
do ix = 1, kx
   do iy = 1, ky
      ftmpy(iy) = data(ix,iy)
   end do
   call fftf1 ( ky, ftmpy )
   do iy = 1, ky
      data(ix,iy) = ftmpy(iy)
   end do
end do
return
end
```

```
      subroutine fftfx2( kx, ky, data )
      dimension data(kx,ky), ftmpx(lx)
!HPF$ PROCESSORS proc(lpara)
!HPF$ DISTRIBUTE (*,BLOCK) ONTO proc :: data
      interface
          extrinsic( ... ) subroutine fftf1( ... )
          ...
          end subroutine
      end interface
!HPF$ INDEPENDENT,NEW(ix,ftmpx)
      do iy = 1, ky
         do ix = 1, kx
            ftmpx(ix) = data(ix,iy)
         end do
         call fftf1(ftmpx, ... , ... )
         do ix = 1, kx
            data(ix,iy) = ftmpx(ix)
         end do
      end do
      return
      end
      subroutine fftfy2( kx, ky, data )
      dimension data(kx,ky), ftmpy(ly)
!HPF$ PROCESSORS proc(lpara)
!HPF$ DISTRIBUTE (BLOCK,*) ONTO proc :: data
      ...
```



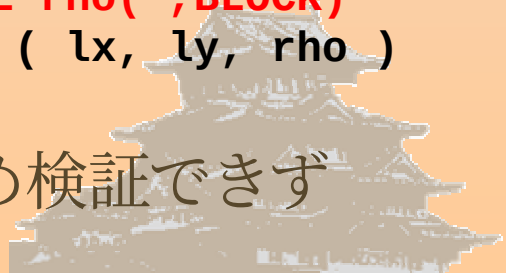
再分散の最適化

- REDISTRIBUTEと記述的分散(descriptive)を用いると，4回の再分散が2回になる.

```
!HPF$ DISTRIBUTE rho(*,BLOCK)
!HPF$ DISTRIBUTE ck(*,BLOCK)
c.... forward FFT
      call fftfx2 ( lx, ly, rho )
c--- remap (*,BLOCK) -> (BLOCK,*)
      call fftfy2 ( lx, ly, rho )
c--- remap (BLOCK,*) -> (*,BLOCK)
c.... form factor
      rho = ck* rho
c.... backward FFT
c--- remap (*,BLOCK) -> (BLOCK,*)
      call fftby2 ( lx, ly, rho )
c--- remap (BLOCK,*) -> (*,BLOCK)
      call fftbx2 ( lx, ly, rho )
```

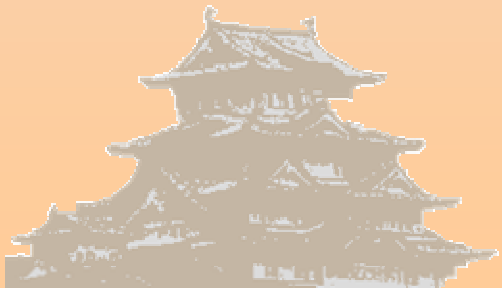
```
!HPF$ DISTRIBUTE rho(*,BLOCK)
!HPF$ DISTRIBUTE ck(BLOCK,*)
c.... forward FFT
      call fftfx2 ( lx, ly, rho )
!HPF$ REDISTRIBUTE rho(BLOCK,*)
      call fftfy2 ( lx, ly, rho )
c.... form factor
      rho = ck* rho
c.... backward FFT
      call fftby2 ( lx, ly, rho )
!HPF$ REDISTRIBUTE rho(*,BLOCK)
      call fftbx2 ( lx, ly, rho )
```

注：未実装のため検証できず



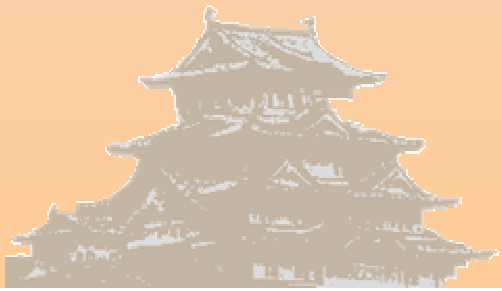
挿入したHPF指示文の行数

オリジナル	1591
修正／追加	68
INTERFACE BLOCK	77
PROCESSORS	11
DISTRIBUTE	19
INDEPENDENT	19
ON-LOCAL/ENDON	12



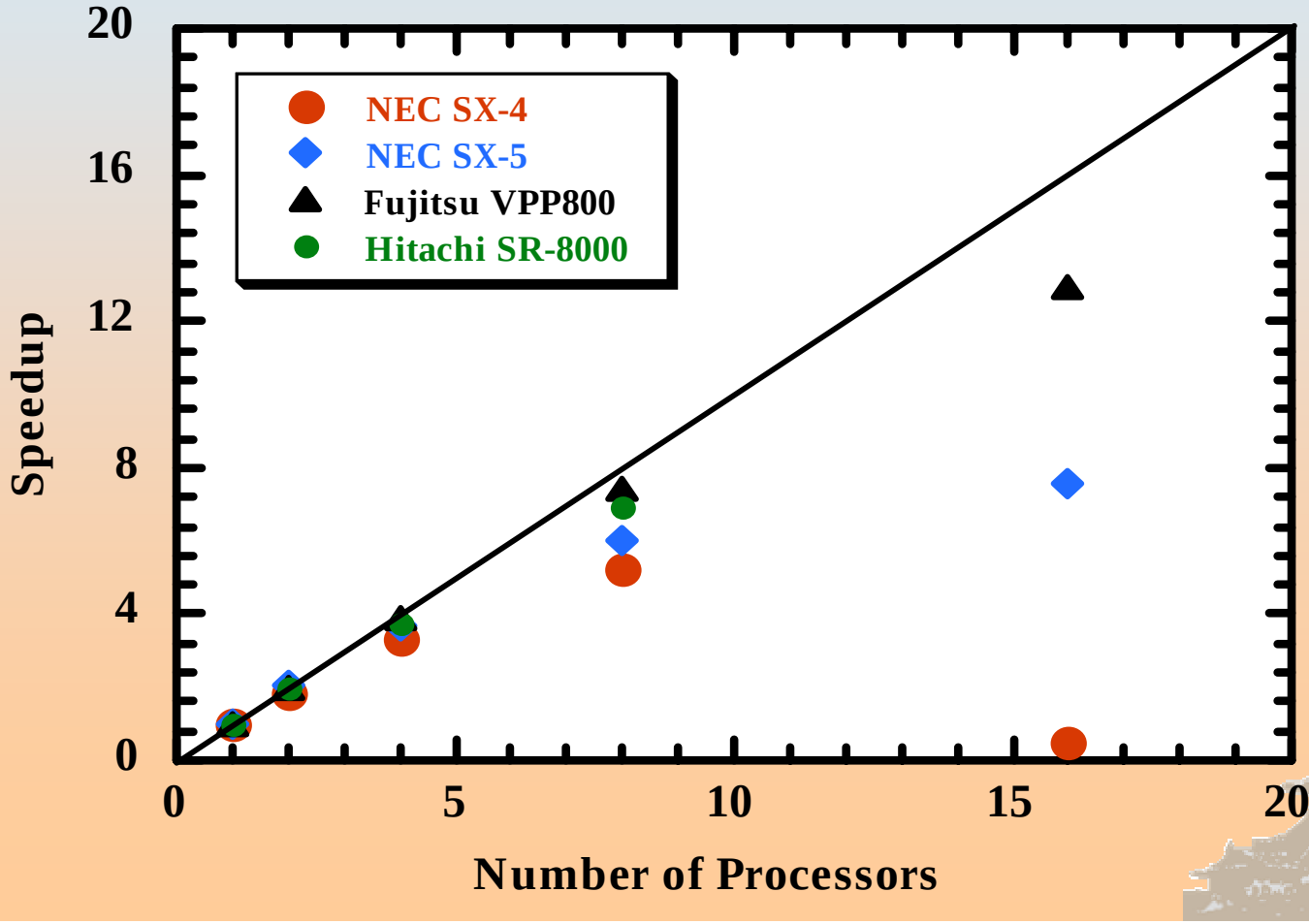
実行時間の比較 秒]

proc#	SX-4	SX-5	VPP800	SR-8000
1	40.25	20.36	24.82	27.65
2	21.43	9.84	12.57	13.84
4	12.09	5.54	6.37	7.33
8	7.66	3.37	3.34	3.97
16	78.09	2.69	1.92	
32			1.36	



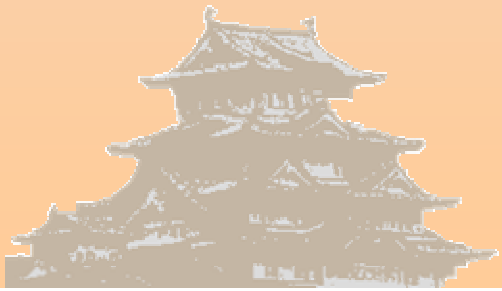
並列化によるスピードアップ

- Lx=Ly=128, nopm=200



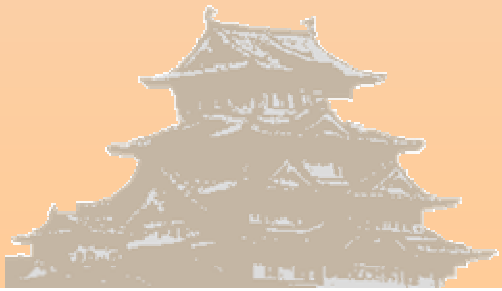
2次元静電粒子コードのまとめ

- 並列化のために，若干のソースコードの修正が必要であった.
- NEC，富士通，日立のHPFコンパイラは，プロセッサ台数が少ないときは，良好な並列性能を実現している.
  - 配列のREDUCTIONが遅い！



SPEC OMP2001

- SPECによって開発されたOpenMPによる共有メモリ型並列計算機システムの性能評価を行うベンチマークプログラムである.
  - HPFと同様に指示文ベースである.
  - 単なるベンチマークコードではなく, 実用的なコードである.
  - FORTRANコード8本とCコード2本からなる.
- SPEC HPFの開発を目指す.



SWIMの並列化

- 浅水方程式による海水のシミュレーションプログラムである.
- 結果を格納する変数の添字が異なるDOループで並列性能が悪い／並列化できない.
  - Owner Computes Rule

```
      dimension a(10),b(9)
!HPF$ PROCESSORS proc(2)
!HPF$ DISTRIBUTE (BLOCK) ONTO proc :: a,b
      do i=1,9
        a(i+1) = ...
        b(i)   = ...
      enddo
```

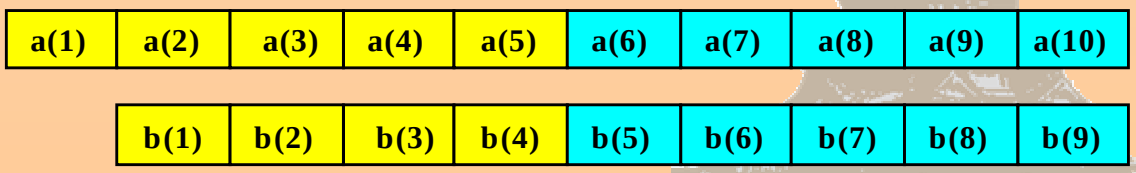
SWIMにおける問題の解決

- DOループの分割による並列化

```
do i =1, 9
  a(i+1) = ...
end do
do i =1, 9
  b(i)   = ...
end do
```

- ALIGN指示文による並列化

```
!HPF$ ALIGN b(i) WITH a(i+1)
      ...
      do i = 1, 9
        a(i+1) = ...
        b(i)   = ...
      end do
```

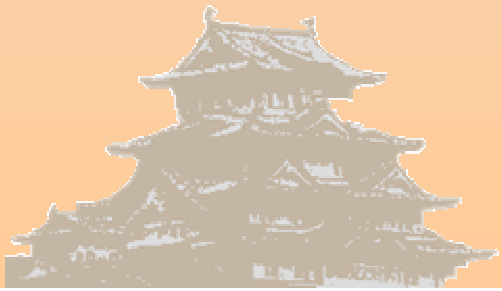


DOループの分割とALIGN指示文の併用

- ベクトル化の効率を考慮する.

```
      ...
!HPF$ DISTRIBUTE (BLOCK)
     &  ONTO proc :: a,p
!HPF$ ALIGN b(i) WITH
     &  a(i+1)
      ...
!HPF$ INDEPENDENT
      do i = 1, 9
       a(i+1) =  p(i)*2
       b(i)   = -p(i)*2
      end do
      ...
!HPF$ INDEPENDENT
      do i = 1, 9
       a(i) =  0
      end do
!HPF$ INDEPENDENT
      do i = 1, 9
       b(i) =  1
      end do
```

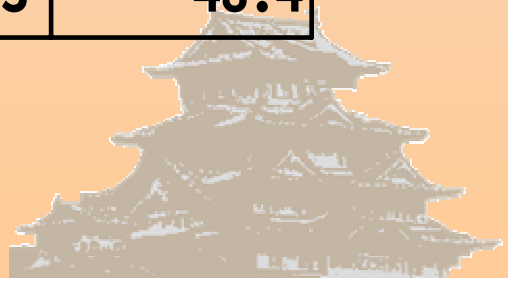
```
      ...
!HPF$ DISTRIBUTE (BLOCK)
     &  ONTO proc :: a,b,p
      ...
!HPF$ INDEPENDENT
      do i = 1, 9
       a(i+1) =  p(i)*2
      end do
!HPF$ INDEPENDENT
      do i = 1, 9
       b(i)   = -p(i)*2
      end do
      ...
!HPF$ INDEPENDENT
      do i = 1, 9
       a(i) =  0
       b(i) =  1
      end do
```



SWIMの実行時間 秒]

- OpenMPと同程度の性能が得られた

		工夫無し	DOの分割		DOの分割とALIGN		OpenMP
		SX-5	SX-5	VPP800	SX-5	VPP800	SX-5
プロセッサ数	1	505.3	314.6	427.6	294.7	357.6	283.2
	2	265.9	165.3	218.3	153.3	176.7	150.4
	3	183.7	114.6	147.1	109.3	121.3	103.2
	4	144.4	89.8	111.5	82.3	92.4	81.4
	8	81.8	52.2	60.7	51.3	52.3	48.4



MGRIDの並列化

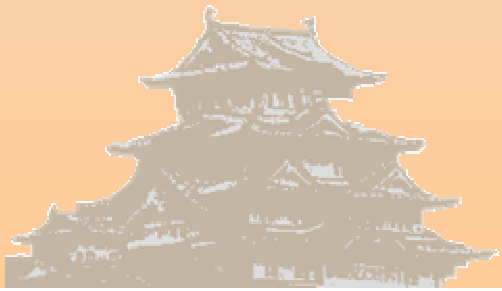
- マルチグリッドにより3次元のポテンシャル場を計算をするプログラムである.
- 主プログラムと副プログラムで引数の次元数が異なるため，並列化することができなかった.

主プログラム

```
dimension big(M),is(k),il(k)
•••
do i = 1, k
  call sub( big(is(k)), il(k) )
end do
•••
stop
end
```

副プログラム

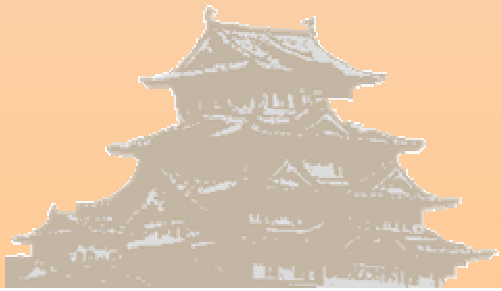
```
subroutine sub( a, m )
dimension a(m,m,m)
•••
return
end
```



MGRIDにおける問題の解決

- 動的な配列領域の確保と値のコピー

```
      subroutine sub(aa,m)
      dimension aa(m*m*m)
!HPF$ PROCESSORS proc(2)
!HPF$ DISTRIBUTE (BLOCK) ONTO proc :: a
      allocatable a(:,:,:)
      allocate(a(m,m,m))
      a  ← aa
      ...
      ...
      aa ← a
      deallocate(a)
      return
      end
```



MGRIDの実行時間［秒］

- 動的な配列確保および配列コピーのオーバーヘッドのため，高い並列性能は得られない.
  - 最近は，主プログラムと副プログラムで引数の次元数が異なるコーディングをする必要性が減っている.

プロセッサ数	HPF	OpenMP
1	285.3	230.2
2	216.2	118.6
3	176.1	81.1
4	160.9	63.1
8	138.8	36.0

＊ SX-5のみ