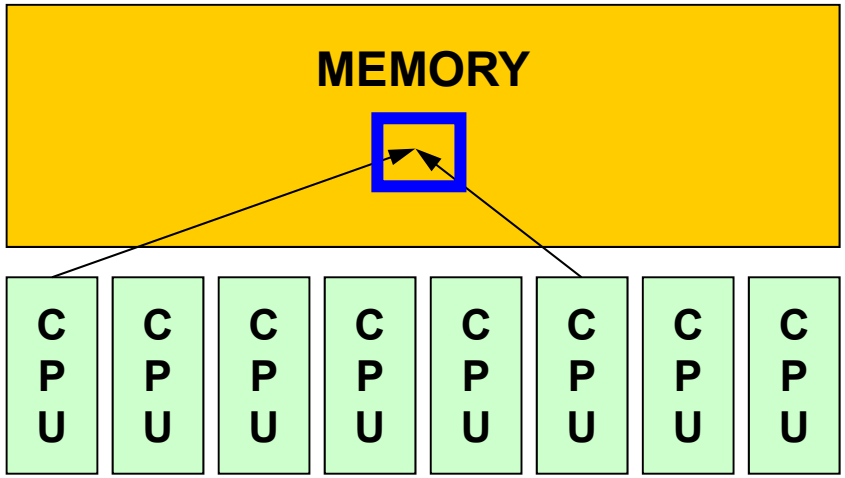
共有メモリ型計算機



- SMP
  - Symmetric Multi Processors
  - 複数のCPU（コア）で同じメモリ空間を共有するアーキテクチュア

OpenMPとは

http://www.openmp.org

- 共有メモリ型並列計算機用のDirectiveの統一規格
  - この考え方が出てきたのは MPIやHPFに比べると遅く1996年であるという。
  - 現在 Ver.4.0
- 背景
  - CrayとSGIの合併
  - ASCI計画の開始
- 主な計算機ベンダーが集まって OpenMP ARBを結成し，1997年にはもう規格案ができていたそうである
  - SC98ではすでにOpenMPのチュートリアルがあったし，すでにSGI Origin2000でOpenMP-MPIハイブリッドのシミュレーションをやっている例もあった。

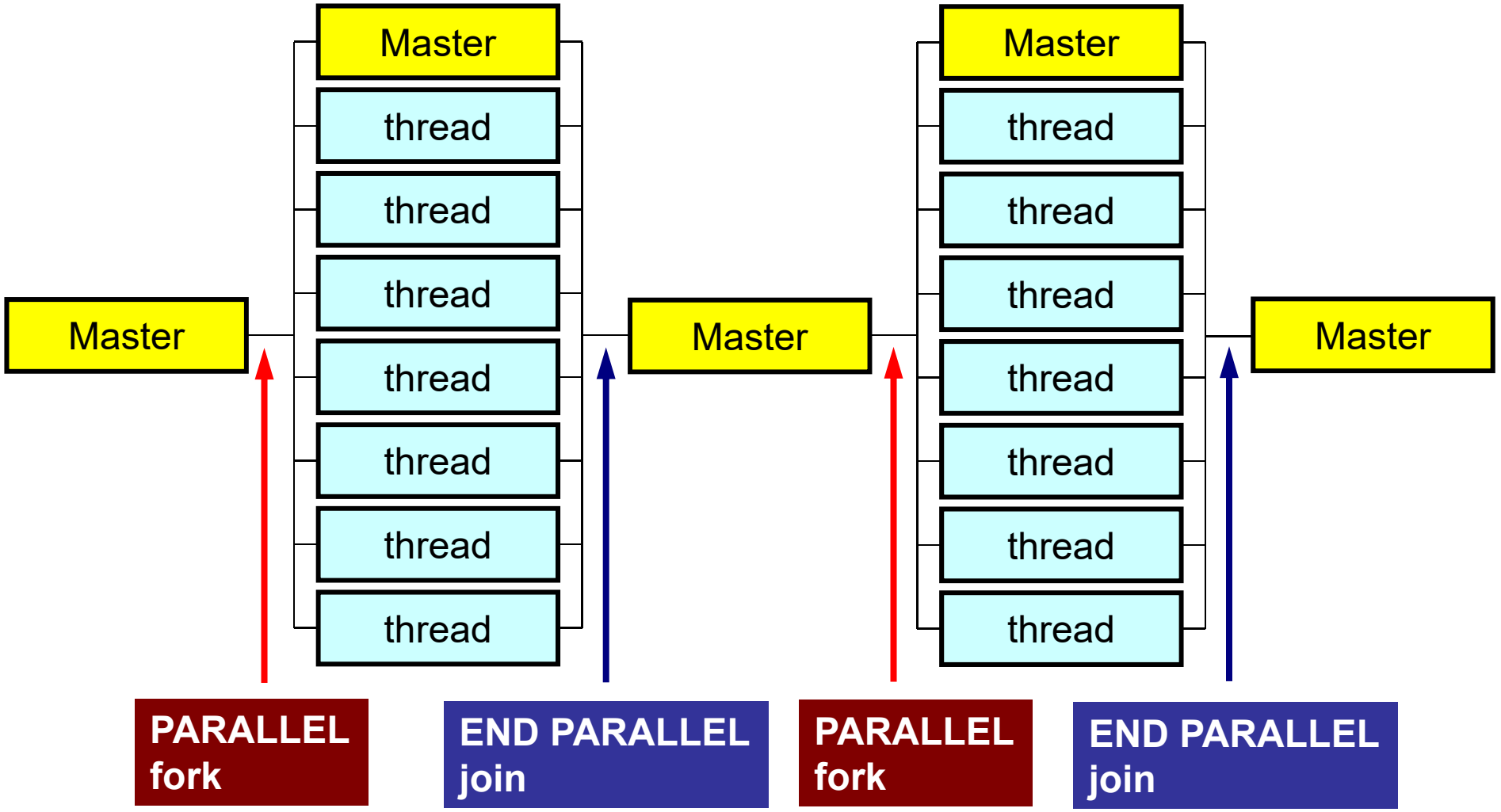
OpenMPとは（続き）

- OpenMPはFortan版とC/C++版の規格が全く別々に進められてきた。
  - Ver.2.5で言語間の仕様を統一
- Ver.4.0ではGPU，Intel-MIC等Co-Processor, Accelerator環境での動作も考慮
  - OpenACC

OpenMPの概要

- 基本的仕様
  - プログラムを並列に実行するための動作をユーザーが明示
  - OpenMP実行環境は，依存関係，衝突，デッドロック，競合条件，結果としてプログラムが誤った実行につながるような問題に関するチェックは要求されていない。
  - プログラムが正しく実行されるよう構成するのはユーザーの責任である。
- 実行モデル
  - fork-join型並列モデル
    - 当初はマスタスレッドと呼ばれる単一プログラムとして実行を開始し，「PARALLEL」，「END PARALLEL」ディレクティヴの対で並列構造を構成する。並列構造が現れるとマスタスレッドはスレッドのチームを生成し，そのチームのマスタとなる。
  - いわゆる「入れ子構造」も可能であるが，ここでは扱わない

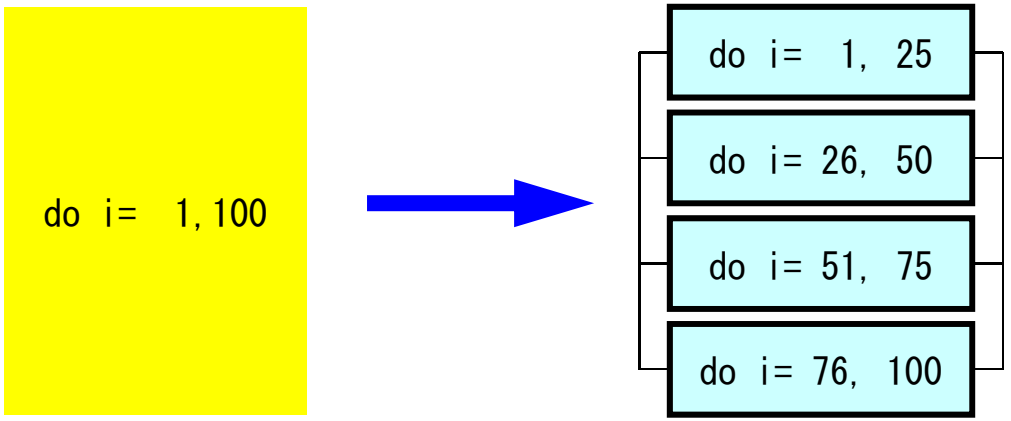
Fork-Join 型並列モデル



スレッド数

- 環境変数 **OMP_NUM_THREADS**
  - 値の変え方
    - bash(.bashrc)　　**export OMP_NUM_THREADS=8**
    - csh(.cshrc)　　**setenv OMP_NUM_THREADS 8**

- **たとえば，OMP_NUM_THREADS=4とすると，以下のように i=1~100のループが4分割され，同時に実行される。**



OpenMPに関連する情報

- OpenMP Architecture Review Board (ARB)
  - http://www.openmp.org
- 参考文献
  - Chandra, R. et al.「Parallel Programming in OpenMP」（Morgan Kaufmann）
  - Quinn, M.J.「Parallel Programming in C with MPI and OpenMP」（McGrawHill）
  - Mattson, T.G. et al.「Patterns for Parallel Programming」（Addison Wesley）
  - 牛島「OpenMPによる並列プログラミングと数値計算法」（丸善）
  - Chapman, B. et al.「Using OpenMP」（MIT Press）
- 富士通他による翻訳：（OpenMP 3.0）必携！
  - http://www.openmp.org/mp-documents/OpenMP30spec-ja.pdf

OpenMPに関する国際会議

- WOMPEI（International Workshop on OpenMP: Experiences and Implementations ）
  - 日本（1年半に一回）
- WOMPAT（アメリカ），EWOMP（欧州）

- 2005年からこれらが統合されて「IWOMP」となる，毎年開催。
  - International Workshop on OpenMP
  - http://www.nic.uoregon.edu/iwomp2005/
  - Eugene, Oregon, USA

OpenMPの特徴

- ディレクティヴ（指示行）の形で利用
  - 挿入直後のループが並列化される
  - コンパイラがサポートしていなければ，コメントとみなされる

OpenMP/Directives Array Operations

Simple Substitution

```
!$omp parallel do
      do i= 1, NP
        W(i,1)= 0.d0
        W(i,2)= 0.d0
      enddo
!$omp end parallel do
```

DAXPY

```
!$omp parallel do
      do i= 1, NP
        Y(i)= ALPHA*X(i) + Y(i)
      enddo
!$omp end parallel do
```

Dot Products

```
!$omp parallel do private(ip,iS,iE,i)
!$omp&             reduction(+:RHO)
      do ip= 1, PEsmpTOT
        iS= STACKmcG(ip-1) + 1
        iE= STACKmcG(ip  )
        do i= iS, iE
          RHO= RHO + W(i,R)*W(i,Z)
        enddo
      enddo
!$omp end parallel do
```

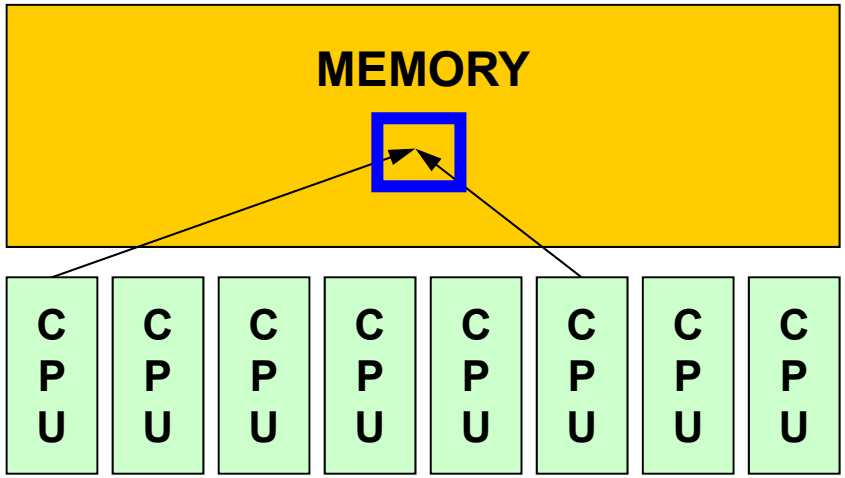
OpenMP/Direceives Matrix/Vector Products

```
!$omp parallel do private(ip,iS,iE,i,j)
      do ip= 1, PEsmpTOT
        iS= STACKmcG(ip-1) + 1
        iE= STACKmcG(ip  )
        do i= iS, iE
          W(i,Q)= D(i)*W(i,P)
          do j= 1, INL(i)
            W(i,Q)= W(i,Q) + W(IAL(j,i),P)
          enddo
          do j= 1, INU(i)
            W(i,Q)= W(i,Q) + W(IAU(j,i),P)
          enddo
        enddo
      enddo
!$omp end parallel do
```

OpenMPの特徴

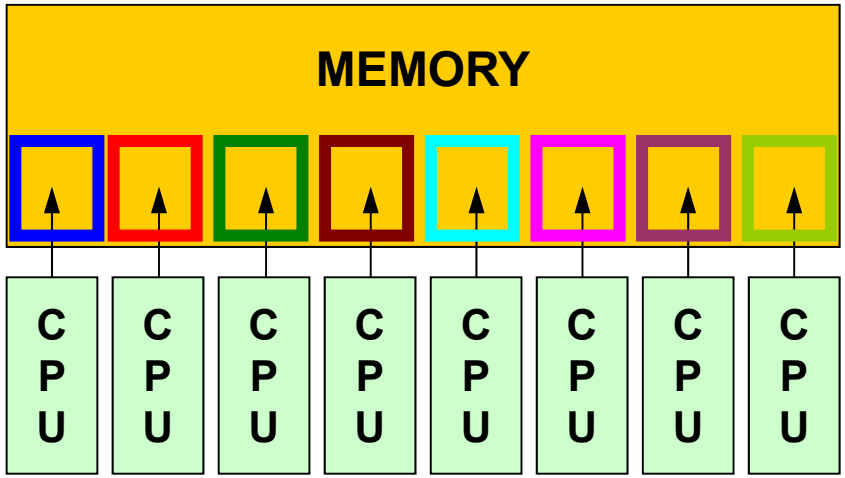
- ディレクティヴ（指示行）の形で利用
  - 挿入直後のループが並列化される
  - コンパイラがサポートしていなければ，コメントとみなされる
- 何も指定しなければ，何もしない
  - 「自動並列化」，「自動ベクトル化」とは異なる。
  - 下手なことをするとおかしな結果になる：ベクトル化と同じ
  - データ分散等（Ordering）は利用者の責任
- 共有メモリユニット内のプロセッサ数に応じて，「Thread」が立ち上がる
  - 「Thread」：MPIでいう「プロセス」に相当する。
  - 普通は「Thread数＝共有メモリユニット内プロセッサ数，コア数」であるが最近のアーキテクチャではHyper Threading（HT）がサポートされているものが多い（1コアで2-4スレッド）

メモリ競合



- 複雑な処理をしている場合，複数のスレッドがメモリ上の同じアドレスにあるデータを同時に更新する可能性がある。
  - 複数のCPUが配列の同じ成分を更新しようとする。
  - メモリを複数のコアで共有しているためこのようなことが起こりうる。
  - 場合によっては答えが変わる

メモリ競合（続き）



- 本演習で扱っている例は，そのようなことが生じないよう，各スレッドが同時に同じ成分を更新するようなことはないようにする。
  - これはユーザーの責任でやること，である。
- ただ多くのコア数（スレッド数）が増えるほど，メモリへの負担が増えて，処理速度は低下する。

OpenMPの特徴（続き）

- 基本は「!omp parallel do」～「!omp end parallel do」
- 変数について，グローバルな変数（shared）と，Thread内でローカルな「private」な変数に分けられる。
  - デフォルトは「shared」
  - 内積を求める場合は「reduction」を使う

```
!$omp parallel do private(ip,iS,iE,i)
!$omp&            reduction(+:RHO)
      do ip= 1, PEsmpTOT
        iS= STACKmcG(ip-1) + 1
        iE= STACKmcG(ip  )
        do i= iS, iE
          RHO= RHO + W(i,R)*W(i,Z)
        enddo
      enddo
!$omp end parallel do
```

W(:,:), R, Z, PEsmpTOT などはグローバル変数

FORTRANとC

```
use omp_lib
...
!$omp parallel do shared(n,x,y) private(i)
      do i= 1, n
        x(i)= x(i) + y(i)
      enddo
!$ omp end parallel do
```

```
#include <omp.h>
...
{
  #pragma omp parallel for default(none) shared(n,x,y) private(i)

    for (i=0; i<n; i++)
        x[i] += y[i];
}
```

本講義における方針

- OpenMPは多様な機能を持っているが，それらの全てを逐一教えることはしない。
  - 講演者も全てを把握，理解しているわけではない。

- 数値解析に必要な最低限の機能のみ学習する。
  - 具体的には，講義で扱っているICCG法によるポアソン方程式ソルバーを動かすために必要な機能のみについて学習する
  - それ以外の機能については，自習，質問のこと(全てに答えられるとは限らない)。
- MPIと同じ

最初にやること

- use omp_lib　　FORTRAN
- #include <omp.h>　　C

- 様々な環境変数，インタフェースの定義（OpenMP3.0以降でサポート）

OpenMPディレクィヴ（FORTRAN）

```
sentinel directive_name [clause[[,] clause]…]
```

- 大文字小文字は区別されない。
- sentinel
  - 接頭辞
  - FORTRANでは「!$OMP」,「C$OMP」,「*$OMP」, 但し自由ソース形式では「!$OMP」のみ。
  - 継続行にはFORTRANと同じルールが適用される。以下はいずれも「!$OMP PARALLEL DO SHARED(A,B,C)」

```
!$OMP PARALLEL DO
!$OMP+SHARED (A,B,C)
```

```
!$OMP PARALLEL DO &
!$OMP SHARED (A,B,C)
```

OpenMPディレクィヴ（C）

```
#pragma omp directive_name [clause[[,] clause]…]
```

- 継続行は「＼」
- 小文字を使用（変数名以外）

```
#pragma omp parallel for shared (a,b,c)
```

PARALLEL DO

```
!$OMP PARALLEL DO[clause[[,] clause] … ]
  (do_loop)
!$OMP END PARALLEL DO
```

```
#pragma parallel for [clause[[,] clause] … ]
  (for_loop)
```

- 多重スレッドによって実行される領域を定義し，DOループの並列化を実施する。
- 並び項目（clause）：よく利用するもの
  - PRIVATE（list）
  - SHARED（list）
  - DEFAULT（PRIVATE|SHARED|NONE）
  - REDUCTION（{operation|intrinsic}: list）

REDUCTION

```
REDUCTION ({operator|instinsic}: list)
reduction ({operator|instinsic}: list)
```

- 「MPI_REDUCE」のようなものと思えばよい
- Operator
  - ＋, *, -, .AND., .OR., .EQV., .NEQV.
- Intrinsic
  - MAX, MIN, IAND, IOR, IEQR

実例A1：簡単なループ

```
!$OMP PARALLEL DO
      do i= 1, N
        B(i)= (A(i) + B(i)) * 0.50
      enddo
!$OMP END PARALLEL DO
```

- ループの繰り返し変数（ここでは「i」）はデフォルトでprivateなので，明示的に宣言は不要。
- 「END PARALLEL DO」は省略可能。
  - C言語ではそもそも存在しない

実例A2：REDUCTION

```
!$OMP PARALLEL DO PRIVATE (i,Alocal,Blocal) REDUCTION(+:A,B)
      do i= 1, N
        Alocal= dfloat(i+1)
        Blocal= dfloat(i+2)
        A= A + Alocal
        B= B + Blocal
      enddo
!$OMP END PARALLEL DO
```

- 「END PARALLEL DO」は省略可能。

OpenMP使用時に呼び出すことのできる関数群

関数名	内容
int omp_get_num_threads (void)	スレッド総数
int omp_get_thread_num (void)	自スレッドのID
double omp_get_wtime (void)	MPI_Wtimeと同じ
void omp_set_num_threads (int num_threads) call omp_set_num_threads (num_threads)	スレッド数設定

OpenMPを適用するには？(内積)

```
VAL= 0.d0
do i= 1, N
  VAL= VAL + W(i,R) * W(i,Z)
enddo
```

OpenMPを適用するには？(内積)

```
      VAL= 0.d0
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
```

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(i) REDUCTION(+:VAL)
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
!$OMP END PARALLEL DO
```

OpenMPディレクティヴの挿入
これでも並列計算は可能

OpenMPを適用するには？(内積)

```
      VAL= 0.d0
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
```

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(i) REDUCTION(+:VAL)
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
!$OMP END PARALLEL DO
```

OpenMPディレクティヴの挿入
これでも並列計算は可能

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(ip,i) REDUCTION(+:VAL)
      do ip= 1, PEsmpTOT
        do i= index(ip-1)+1, index(ip)
          VAL= VAL + W(i,R) * W(i,Z)
        enddo
      enddo
!$OMP END PARALLEL DO
```

多重ループの導入
PEsmpTOT：スレッド数
あらかじめ「INDEX(:)」を用意しておく
より確実に並列計算実施
（別に効率がよくなるわけでは無い）

OpenMPを適用するには？(内積)

```
      VAL= 0.d0
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
```

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(i) REDUCTION(+:VAL)
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
!$OMP END PARALLEL DO
```

OpenMPディレクティヴの挿入
これでも並列計算は可能

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(ip,i) REDUCTION(+:VAL)
      do ip= 1, PEsmpTOT
        do i= index(ip-1)+1, index(ip)
          VAL= VAL + W(i,R) * W(i,Z)
        enddo
      enddo
!$OMP END PARALLEL DO
```

多重ループの導入
PEsmpTOT：スレッド数
あらかじめ「INDEX(:)」を用意しておく
より確実に並列計算実施

PEsmpTOT個のスレッドが立ち上がり，
並列に実行

OpenMPを適用するには？(内積)

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(ip,i) REDUCTION(+:VAL)
      do ip= 1, PEsmpTOT
        do i= index(ip-1)+1, index(ip)
          VAL= VAL + W(i,R) * W(i,Z)
        enddo
      enddo
!$OMP END PARALLEL DO
```

多重ループの導入
PEsmpTOT：スレッド数
あらかじめ「INDEX(:)」を用意しておく
より確実に並列計算実施

PEsmpTOT個のスレッドが立ち上がり，並列に実行

各要素が計算されるスレッドを指定できる

GPUには良くないらしい

例えば，N=100，PEsmpTOT=4とすると：

INDEX(0)= 0
INDEX(1)= 25
INDEX(2)= 50
INDEX(3)= 75
INDEX(4)= 100

課題

- CGソルバー（solver_CG，solver_SR）のOpenMPによるマルチスレッド化，Hybrid並列化
- 行列生成部（mat_ass_main，mat_ass_bc）のマルチスレッド化，Hybrid並列化

- 問題サイズを変更して計算を実施してみよ
- Hybridでノード内スレッド数を変化させてみよ
  - OMP_NUM_THREADS

FORTRAN(solver_CG)

```
!$omp parallel do private(i)
      do i= 1, N
         X(i)  = X (i)    + ALPHA * WW(i,P)
        WW(i,R)= WW(i,R) - ALPHA * WW(i,Q)
      enddo
```

```
      DNRM20= 0.d0
!$omp parallel do private(i) reduction (+:DNRM20)
      do i= 1, N
        DNRM20= DNRM20 + WW(i,R)**2
      enddo
```

```
!$omp parallel do private(j,k,i,WVAL)
      do j= 1, N
        WVAL= D(j)*WW(j,P)
        do k= index(j-1)+1, index(j)
           i= item(k)
          WVAL= WVAL + AMAT(k)*WW(i,P)
        enddo
        WW(j,Q)= WVAL
      enddo
```

C(solver_CG)

```
#pragma omp parallel for private (i)
    for(i=0;i<N;i++){
      X [i]    +=  ALPHA *WW[P][i];
      WW[R][i]+= -ALPHA *WW[Q][i];
    }

    DNRM20= 0.e0;
#pragma omp parallel for private (i) reduction (+:DNRM20)
    for(i=0;i<N;i++){
      DNRM20+=WW[R][i]*WW[R][i];
    }

#pragma omp parallel for private (j,i,k,WVAL)
    for( j=0;j<N;j++){
      WVAL= D[j] * WW[P][j];
      for(k=indexLU[j];k<indexLU[j+1];k++){
        i=itemLU[k];
        WVAL+= AMAT[k] * WW[P][i];
    }
    WW[Q][j]=WVAL;
```