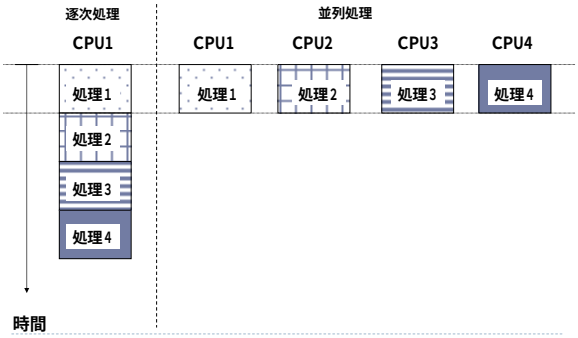
OpenMP基礎

岩下 武史 (学術情報メディアセンター)

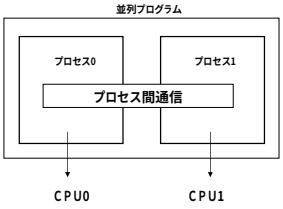
2009/9/4

並列処理とは



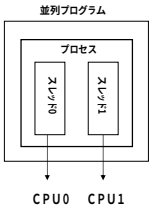
2種類の並列処理方法

- プロセス並列



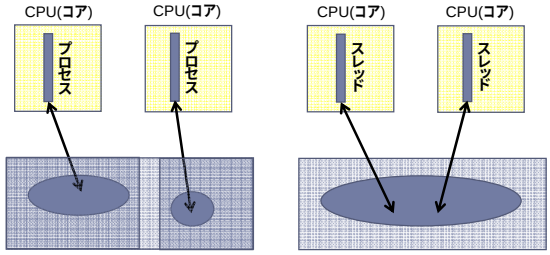
・分散メモリ型並列計算機向け (PCクラスタ)
(共有メモリ型でも使用可能)
・メッセージパッシングライブラリ(MPIなど)を用いる

◆ スレッド並列



・共有メモリ型並列計算機向け (HX600ノード内)
・OpenMPを用いる
・コンパイラによる自動並列化もある

プロセス並列とスレッド並列



HX600における並列処理

- **ノード内での並列処理**
  - プロセス並列、スレッド並列のいずれも可能
- **複数ノードでの並列処理**
  - プロセス並列の利用が必須
  - プロセス並列のみを利用
    - Flat-MPI
  - プロセス／スレッド併用並列処理
    - **ハイブリッド並列処理**
    - MPI＆**自動並列**, MPI & OpenMP

OpenMPとは

- **共有メモリ型並列計算機における並列プログラミングの統一規格**
- **並列実行単位は “スレッド”**

- **プログラミング**
  - **並列化を指示する指示行をプログラムに挿入**

- **規格** (http://www.openmp.org **参照**)
  - OpenMP Fortran Application Program Interface Version2.0 (2000.11)
  - OpenMP C/C++ Program Interface Version2.0 (2002.3)
  - OpenMP Application Program Interface Version 2.5 (2005.5)
  - OpenMP Application Program Interface Version 3.0 (2008.5)
    - C, C++, FORTRAN**の全てを含む**

OpenMPプログラミングの解説

- **指示文（ディレクティブ）の形式**
- **マルチスレッドでの並列実行**
- Work-Sharing**構造**
- **変数の属性**
- **同期**
- **まとめ**

ディレクティブの形式

- **特別なコメント**
  - OpenMP**コンパイラが解釈**
  - Fortran**コンパイラではただのコメント**
- **形式**
  - **固定形式** : !OMP ,COMP, *$OMP
  - **自由形式** : !$OMP
  - ✓ **例** ： !$OMP PARALLEL
- **継続行**
  - ✓ !$OMP PARALLEL DO REDUCTION(+:x) **を2行に継続**
  - **固定形式** !$OMP PARALLEL DO
    !$OMP+REDUCTION(+:x)
  - **自由形式**
    !$OMP PARALLEL DO &
    !$OMP REDUCTION(+:x)

ディレクティブの形式（C言語）

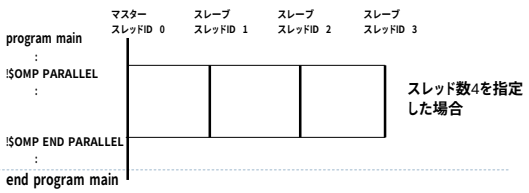
- **形式**
  - #pragma omp **指示子 構造ブロック**
  - ✓ **例** ： #pragma omp parallel
    { …
    }
  - ✓ **例2** ： #pragma omp for

コンパイラによるディレクティブの解釈

- **コンパイルオプションにより指定**
  - **指定**ON －＞ OpenMP**のディレクティブとして解釈**
  - **指定**OFF －＞ **コメントとして無視する**
  - **指定**（ON）**例**:
    - HX600**上** frt –KOMP ***.f90
    - Linux PC (x86) Intel **コンパイラ** ifort –openmp ***.f90
- **生成された実行バイナリ**
  - **環境変数**（OMP_NUM_THREADS）**により指定されたスレッド数での並列実行が行われる**
    - **プログラム内で指定する方法もある** omp_set_num_threads()**関数を利用**

マルチスレッドでの実行イメージ

- **プログラム実行開始時はマスタースレッドのみ**
- PARALLEL**ディレクティブによりスレーブスレッドを生成**
  - **スレッドID： マスタースレッドは0、スレーブスレッドは1～**
  - **チーム： 並列実行を行うスレッドの集団**
  - **スレッド生成後、全てのスレッドで冗長実行**
- END PARALLEL**ディレクティブによりスレーブスレッドが消滅**



OpenMPによる並列化プログラムの基本構成例 （Fortran90プログラム）

```
program main
integer :: i,j
double precision :: a,b
…
!$OMP PARALLEL
…
…
…
!$OMP END PARALLEL
…
…
end
```

複数のスレッドにより並列実行される部分

OpenMPによる並列化プログラムの基本構成例（Cプログラム）

```
int main(){
int i,j;
double a,b;
…
#pragma omp parallel
{
…
…
}
}
```

⇒ **複数のスレッドにより並列実行される部分**

複数スレッドによる冗長実行

```
program main
integer :: i,j
double precision :: a,b
…
!$OMP PARALLEL
   a=b
!$OMP END PARALLEL
…
…
end
```

OpenMPによる並列化プログラムでは特に何も指示しないと、パラレルリージョン（並列実行される部分）での実行文は冗長実行される

共有メモリなので、複数のスレッドから見て変数aのメモリ上の物理的な番地（実体）は同じ

a=bがスレッドの数だけ行われる

計算の並列化（Work-Sharing構造）

- **チーム内のスレッドに仕事（Work）を分割(Share)する。**
- **Work-Sharing構造の種類**
  - **DOループを各スレッドで分割** (!$OMP DO , !$OMP END DO)
  - **別々の処理を各スレッドが分担**(!$OMP SECTIONS, !$OMP END SECTIONS)
  - **1スレッドのみ実行**(!$OMP SINGLE, !$OMP END SINGLE)
  - **マスタスレッドでのみ実行**(!$OMP MASTER, !$OMP END MASTER)

OMP　DO　（1）

```
Integer :: i
double precision :: a(100), b(100)
!$OMP PARALLEL
!$OMP DO
do i=1,100
   b(i)=a(i)
enddo
!$OMP END DO
!$OMP END PARALLEL
end
```

⇒ **直後のdoループを複数のスレッドで分割して実行せよ　という　指示**

2スレッドの場合：
スレッド0
```
do i=1,50
   b(i)=a(i)
enddo
```
スレッド1
```
do j=51,100
   b(j)=a(j)
enddo
```


OMP　DO　（2）

注意） !$OMP DO **はdoループの中身が並列実行可能かどうかは関知せず、必ず分割してしまう。**

```
Integer :: i
double precision :: a(100), b(0:100)
!$OMP PARALLEL
!$OMP DO
do i=1,100
   b(i)=a(i)+b(i-1)
enddo
!$OMP END DO
!$OMP END PARALLEL
end
```

2スレッドの場合：
スレッド0
```
do i=1,50
   b(i)=a(i)+b(i-1)
enddo
```
スレッド1
```
do j=51,100
   b(j)=a(j)+b(j-1)
enddo
```
⇒ **b(50)の結果がないと本来実行できない**

OMP　DO　（3）

分割を規定する

```
Integer :: i
double precision :: a(100), b(0:100)
!$OMP PARALLEL
!$OMP DO SCHEDULE(STATIC,4)
do i=1,100
   b(i)=a(i)+b(i-1)
enddo
!$OMP END DO
!$OMP END PARALLEL
end
```

⇒ **1～100を4つづつのchunkにわけて、それをサイクリックに各スレッドに割り当てる**

4スレッド実行時
マスタスレッド担当行：
1,2,3,4,17,18,19,20,

OMP Sections

- !$OMP Sections
- !$OMP Section
  計算1　（スレッド0）
- !$OMP Section
  計算2　（スレッド1）
- !$OMP Section
  計算3　（スレッド2）
- !$OMP END Sections

Section毎にスレッドに仕事が割り当てられる

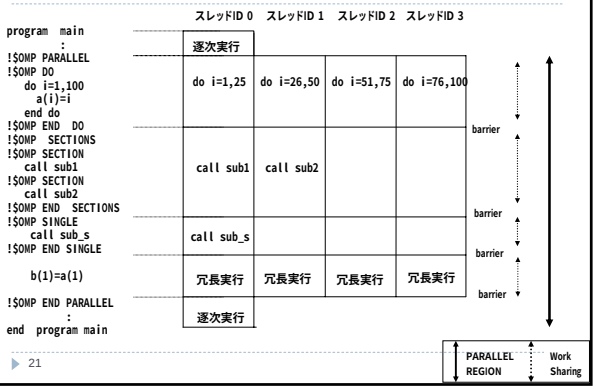
Sectionの数よりもスレッド数が多い場合には仕事をしないスレッドが発生する

OMP Single

- !$OMP Parallel
  並列処理
- !$OMP Single
  逐次処理 → 一つのスレッドのみが処理を行う（冗長実行を防ぐ）
- !$OMP END Single

  並列処理
- !$OMP END Parallel

Work Sharing 実行イメージ



複合パラレルWork-Sharing構造

- **記述方法のひとつ**
- **例**

```
!$OMP PARALLEL
!$OMP DO
    do i=1,n
      :
    end do
!$OMP END DO
!$OMP END PARALLEL
```

⇨

```
!$OMP PARALLEL DO
   do i=1,n
     :
   end do
!$OMP END PARALLEL DO
```

（!$OMP END PARALLEL DOは省略可）

- 同様に以下も記述できる
  - !$OMP PARALLEL SECTIONS

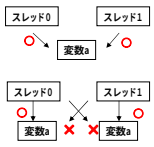
C言語によるWork-Sharing構造の記述

```
int i;
double a[100], b[100];
#pragma omp parallel
{
#pragma omp for
for(i=0; i<100; i++){
b[i]=a[i];
                    }
}
```

```
int i, j, k;
#pragma omp parallel
{
#pragma omp sections
  {
#pragma omp section
            {     j=i;     }
#pragma omp section
            {     k=i;     }
  }
}
```


変数の属性

- **変数属性は大きくわけて次の2つ**
  - SHARED**属性**
    - **プログラムで1つの領域**
    - **どのスレッドからでも参照、更新可能**
  - PRIVATE**属性**
    - **スレッド毎に独立した領域**
    - **各スレッドからだけ参照、更新可能**



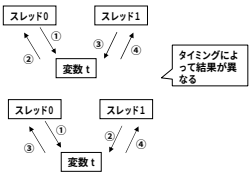
- **デフォルトは基本的にSHARED属性**
- OMP PARALLEL**や**OMP DO，OMP SECTIONS**でprivate変数を指定可能**
  - **それぞれ並列リージョンやWork-sharing構造内でプライベート化　（範囲外では不定の値）**

PRIVATE属性であるべき変数

- **プログラム例**

```
!$OMP PARALLEL DO
  do i=1,n
     t = i + 1
     a(i) = t + n
  end do
!$OMP END PARALLEL DO
```

変数tがshared属性だと…



変数tはPRIVATE属性でなくてはならない

属性の宣言と有効範囲

```
!$OMP PARALLEL DO PRIVATE(t)
  do i=1,n
     t = i + 1
     a(i) = t + n
  end do
!$OMP END PARALLEL DO
  write(*,*) t     ! 不定
```

DO ループの制御変数 i はデフォルトでPRIVATE属性

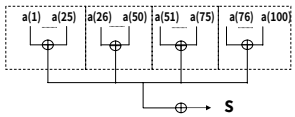
この範囲で t はPRIVATE属性

PRIVATE属性の変数は、有効範囲の外では不定

REDUCTION属性

```
   s = 0
!$OMP PARALLEL DO REDUCTION(+:s)
   do i = 1, 100
      s = s + i
   end do
!$OMP END PARALLEL DO
   write(*,*) s      ! 5050
```

各スレッドで部分和を求めて、最後に加算



- **使用可能な演算子**(operator)**と組み込み関数**(intrinsic)
  - operator : + , - , * , .and. , .or.
  - intrinsic : max , min , iand , ior , ieor
- **形式**
  - REDUCTION({ operator | intrinsic} : **変数名**)

簡単なプログラム例(行列積)

```
program matmul
  integer :: i,j,k
  integer,parameter  :: n=1000
  real :: a(n,n),b(n,n),c(n,n)

call init(a,b,c)       ! 逐次実行

!$OMP PARALLEL DO PRIVATE(k,i)
  do j=1,n                          ! jループを分割して並列実行
     do k=1,n                       !
        do i=1,n                    !
           c(i,j)=c(i,j)+a(i,k)*b(k,j)  !
        end do                      !
     end do                         !
  end do                            !
!$OMP END PARALLEL DO

  write(*,*) "c(1,1)=",c(1,1)       ! 逐次実行

end program matmul
```


簡単なプログラム例(円周率の計算)

```
 program calculate_pi
   integer :: i, n
   real(8) :: w, gsum, pi, v

   n=2000000000              ! 逐次実行
   w = 1.0d0 / n             ! 逐次実行
   gsum = 0.0d0              ! 逐次実行

!$OMP PARALLEL DO PRIVATE(v)  REDUCTION(+:gsum)
    do i = 1, n                          ! iループを分割して並列実行
      v = (real(i,8) - 0.5d0 ) * w       ! 総和演算
      v = 4.0d0 / (1.0d0 + v * v)        !
      gsum = gsum + v                    !
    end do                               !
!$OMP END PARALLEL DO
     pi = gsum * w
     write(*,*) "PI is ", pi
 end program calculate_pi
```


サブルーチンでの変数の属性

```
 program main
   integer ,parameter ::n =100
   integer ::i
   real :: a(n),x
   a=1.0
!$OMP PARALLEL DO PRIVATE(x) REDUCTION(+:y)
   do i=1,n
      call sub0(a,i,x)
      y=y+x
   end do
!$OMP END PARALLEL DO
   write(*,*) y
 end program main

 subroutine sub0(a,i,x)
   integer ,parameter ::n=100
   integer ::i
   real :: a(n),x
   real :: tmp
   tmp=a(i)+1
   x=tmp
   return
 end subroutine sub0
```

(1)引数の変数の属性は受け継がれる
(2)サブルーチン内で定義された変数は PRIVATE属性
(3)大域変数はSHARED属性
(4)SAVE属性をもつ変数は、SHARED属性

なお、(3),(4)はthreadprivate指示文により PRIVATE属性にすることもできる

同期と制御

- **バリア同期**
  - **チーム内のスレッドの到達を待つ**
  - **暗黙のバリア同期**
    - !$OMP END PARALLEL，Work-Sharing**構文の後ろ**
  - **陽に指定** !$OMP BARRIER
  - **バグを作らないために積極的に活用する**
- **複数スレッド間で**shared**変数のアクセス制御**
  - !$OMP CRITICAL, !$OMP END CRITICAL
    - CRITICAL**セクションにはひとつのスレッドしか入れない**
  - !$OMP ATOMIC
    - **直後の実行文の左辺の変数に対するアクセスが逐次化**
  - **並列性能がでないのでなるべく使わないことが望ましい**

OMP Barrier

```
integer,parameter :: num=10
integer :: a(num),b(num)

!$OMP PARALLEL
  do i=1,num
    a(i)=0.0
  enddo

!$OMP BARRIER

!$OMP DO
  do i=1,num
    b(i)=a(i)+1.0
  end do

!$OMP END DO
!$OMP END PARALLEL
```


!$OMP DOの前に暗黙の同期はとられない

明示的にバリア同期を指示

より高度な並列処理（MPI的なプログラム）

- omp_get_num_threads**関数　総使用スレッド数を得る**
- omp_get_thread_num**関数　　スレッド**ID**を得る**

```
integer :: omp_get_thread_num,omp_get_num_threadsnum
external omp_get_thread_num,omp_get_num_threadsnum

!$OMP PARALLEL
      numprocs=omp_get_num_threads()
      myid=omp_get_thread_num()
!$OMP END PARALLEL
!$OMP PARALLEL
      if (myid.eq.0) then
        ・・・
      elseif (myid.eq.1) then
        ・・・
      elseif (myid.eq.2) then
        ・・・
        ・
!$OMP END PARALLEL
  end
```


並列化のポイント　まとめ

- **どの部分が並列化できるのか**
- **変数属性の変更は必要か**
- **同期が必要か**

その他

- **自動並列化機能の利用**
- **コンパイル･実行方法など**
- **参考資料**

コンパイルと実行方法（HX600）

- **コンパイル**
  - –KOMP **オプションをつける**

```
% frt –KOMP samp-omp.f       (Fortran)
% fcc –KOMP samp-omp.c       (C)
% FCC –KOMP samp-omp.C       (C++)
```

- **実行**
  - **次の環境変数を指定する**
    - OMP_NUM_THREADS ：**スレッド数を指定する。**

実行例

- **会話型**

% setenv OMP_NUM_THREADS 4　→ **スレッド数を4に指定する**

% ./a.out

- NQS**バッチ型 サンプルスクリプト**

```
#  Sample script    sample.sh
#   #以降はコメントです。
#   # @$以降はNQSのオプションが指定できます。
#
# @$-eo              # 標準出力標準エラー出力をまとめる
# @$-lP 1            # プロセス数は1
# @$-lp 4            # スレッド数は4
set -x
OMP_NUM_THREADS=4; export OMP_NUM_THREADS
./a.out
#  ---------------------------------------
```


実行例(続き)

- NQS**バッチ型 (続き)**
  - **ジョブの投入**　% qsub –q eh sample.sh
  - **ジョブの確認**　% qstat
  - **ジョブのキャンセル**　% qdel <Request ID>
    (Request ID**は**qstat**コマンドで確認できる**)
  - **結果ファイル**　Nxxxxxx.xxxxx

参考資料など

- 「Parallel Programming in OpenMP」
  - MORGAN KAUFMANN PUBLISHERS
  - ISBN 1-55860-671-8
- **牛島省：「OpenMPによる並列プログラミングと数値計算法」，丸善**
- **OpenMPホームページ**
  http://www.openmp.org/
  - **言語仕様書、サンプルプログラムなど。**
- **オンラインマニュアル**
  - Fortran**使用手引書**
  - C **言語使用手引書**
  - C++**言語使用手引書**

  https://web.kudpc.kyoto-u.ac.jp/