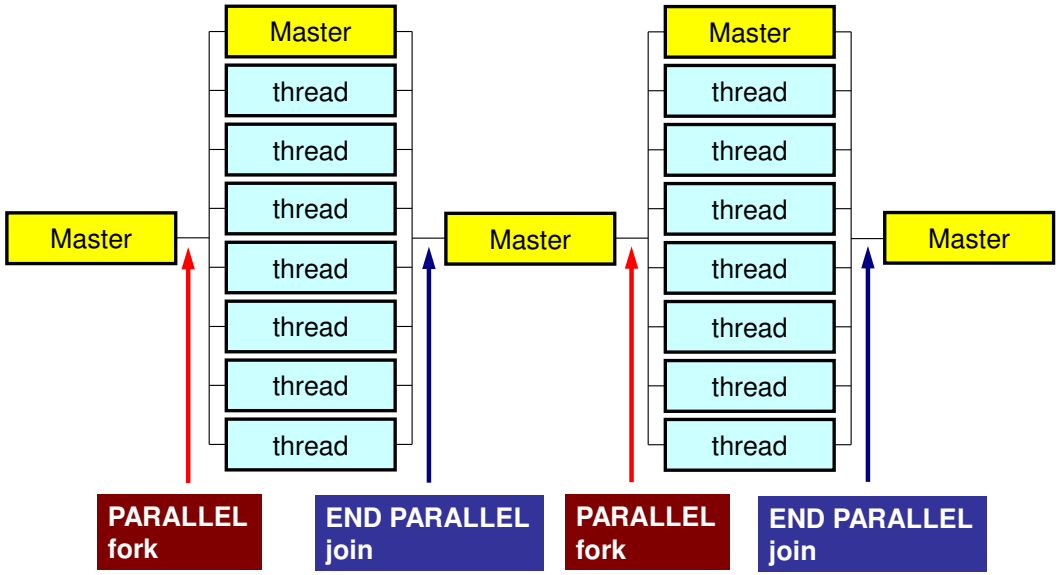
並列有限要素法による
三次元定常熱伝導解析プログラム
OpenMP＋ハイブリッド並列化(2/2)
Fortran編

中島 研吾
東京大学情報基盤センター

omp parallel (do)



- omp parallel-omp end parallelはそのたびにスレッドを生成，消滅させる：fork-join
- ループが連続するとオーバーヘッドになる。
- omp parallel + omp do/omp for

```
!$omp parallel ...

!$omp do
   do i= 1, N
...
!$omp do
   do i= 1, N
...
!$omp end parallel   essential
```

```
#pragma omp parallel {...

#pragma omp for {

...
#pragma omp for {
...

}
```

SOLVER_CG (0/5): Additional Array

```
!$omp parallel do …
do i= 1, N
(…)
enddo
```

```
!$omp parallel

ip= omp_get_thread_num() + 1
do  i= SMPindex(ip-1)+1, SMPindex(ip)
(…)
Enddo

!$omp end parallel
```

PEsmpTOT: Total Number of Threads

Mod.A: src3

```
!$omp parallel
!$omp master
      PEsmpTOT= omp_get_num_threads()
!$omp end master
!$omp end parallel

      allocate (SMPindex(0:PEsmpTOT))
      allocate (W_RHO0   (PEsmpTOT),
     &          W_C10    (PEsmpTOT),
     &          W_BNRM20(PEsmpTOT),
     &          W_DNRM20(PEsmpTOT))

      SMPindex(0)= 0
      nth= N/PEsmpTOT
      nr = N - Nth*PEsmpTOT

      do ip= 1, PEsmpTOT
        SMPindex(ip)= nth
        if (ip.le.nr) SMPindex(ip)= nth + 1
      enddo

      do ip= 1, PEsmpTOT
        SMPindex(ip)= SMPindex(ip-1) +
     &                SMPindex(ip)
      enddo
```

SOLVER_CG (1/5)

Original: src2

```
      do iter= 1, MAXIT
!C
!C +---------------+
!C | {z}= [Minv]{r} |
!C +---------------+
!C===

!$omp parallel do private(i)
      do i= 1, N
        WW(i,Z)= WW(i,R) * WW(i,DD)
      enddo
!C===

!C
!C +---------------+
!C | {RHO}= {r}{z} |
!C +---------------+
!C===
      RHO0= 0.d0

!$omp parallel do private(i) reduction (+:RHO0)
      do i= 1, N
        RHO0= RHO0 + WW(i,R)*WW(i,Z)
      enddo

      call MPI_Allreduce (RHO0, RHO, …)
!C===
```

Mod.A: src3

```
      do iter= 1, MAXIT
!C
!C +---------------+
!C | {z}= [Minv]{r} |
!C +---------------+
!C===

!$omp parallel private(ip,i)
      ip= omp_get_thread_num() + 1
      do i= SMPindex(ip-1)+1, SMPindex(ip)
        WW(i,Z)= WW(i,R) * WW(i,DD)
      enddo
!C===

!C
!C +---------------+
!C | {RHO}= {r}{z} |
!C +---------------+
!C===
      W_RHO0(ip)= 0.0d0
      do i= SMPindex(ip-1)+1, SMPindex(ip)
        W_RHO0(ip)= W_RHO0(ip) + WW(i,R)*WW(i,Z)
      enddo
!$omp end parallel

      RHO0= 0.d0
      do ip0= 1, PEsmpTOT
        RHO0= RHO0 + W_RHO0(ip0)
      enddo
      call MPI_Allreduce (RHO0, RHO, …)
!C===
```

NOT parallel

SOLVER_CG (2/5)

Original: src2

```
!C
!C +-------------------------+
!C | {p} = {z} if      ITER=1    |
!C | BETA= RHO / RHO1  otherwise |
!C +-------------------------+
!C===
      if ( ITER.eq.1 ) then

!$omp parallel do private(i)
        do i= 1, N
          WW(i,P)= WW(i,Z)
        enddo
       else
         BETA= RHO / RHO1
!$omp parallel do private(i)
         do i= 1, N
           WW(i,P)= WW(i,Z) + BETA*WW(i,P)
         enddo
      endif
!C===
```

Mod.A: src3

```
!C
!C +-------------------------+
!C | {p} = {z} if      ITER=1    |
!C | BETA= RHO / RHO1  otherwise |
!C +-------------------------+
!C===

!$omp parallel private(ip,i)
      ip= omp_get_thread_num() + 1
      if ( ITER.eq.1 ) then
        do i= SMPindex(ip-1)+1, SMPindex(ip)
          WW(i,P)= WW(i,Z)
        enddo
       else
         BETA= RHO / RHO1
        do i= SMPindex(ip-1)+1, SMPindex(ip)
          WW(i,P)= WW(i,Z) + BETA*WW(i,P)
        enddo
      endif
!$omp end parallel
!C===
```

SOLVER_CG (3/5)

Original: src2

```
!C
!C +-------------+
!C | {q}= [A]{p} |
!C +-------------+
!C===
!C
!C-- INTERFACE data EXCHANGE

      call SOLVER_SEND_RECV (…)

!$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===
```

Mod.A: src3

```
!C
!C +-------------+
!C | {q}= [A]{p} |
!C +-------------+
!C===
!C
!C-- INTERFACE data EXCHANGE

      call SOLVER_SEND_RECV (…)

!$omp parallel private(ip,ip0,i,j,k,WVAL)
      ip= omp_get_thread_num() + 1

      do j= SMPindex(ip-1)+1, SMPindex(ip)
        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===
```

“!$omp parallel” is still active (fork)

SOLVER_CG (4/5)

Original: src2

```
!C
!C +-------------------+
!C | ALPHA= RHO / {p}{q} |
!C +-------------------+
!C===
      C10= 0.d0
!$omp parallel do private(i) reduction (+:C10)
      do i= 1, N
        C10= C10 + WW(i,P)*WW(i,Q)
      enddo
      call MPI_Allreduce (C10, C1,…)

      ALPHA= RHO / C1
!C===
```

Mod.A: src3

```
!C
!C +-------------------+
!C | ALPHA= RHO / {p}{q} |
!C +-------------------+
!C===
      W_C10(ip)= 0.0d0
      do i= SMPindex(ip-1)+1, SMPindex(ip)
        W_C10(ip)= W_C10(ip) + WW(i,P)*WW(i,Q)
      enddo
!$omp end parallel

      C10= 0.d0
      do ip0= 1, PEsmpTOT
         C10= C10 + W_C10(ip0)
      enddo

      call MPI_Allreduce (C10, C1, …)

      ALPHA= RHO / C1
!C===
```

NOT parallel

SOLVER_CG (5/5)

Original: src2

```
!C
!C +--------------------+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +--------------------+
!C===

!$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
      call MPI_Allreduce (DNRM20, DNRM2,…)

      RESID= dsqrt(DNRM2/BNRM2)
      RHO1 = RHO
…
      enddo
```

Mod.A: src3

```
!C
!C +--------------------+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +--------------------+
!C===

!$omp parallel private(ip,ip0,i)
      ip= omp_get_thread_num() + 1
      do i= SMPindex(ip-1)+1, SMPindex(ip)
         X(i)  = X (i)   + ALPHA * WW(i,P)
        WW(i,R)= WW(i,R) - ALPHA * WW(i,Q)
      enddo

      W_DNRM20(ip)= 0.0d0
      do i= SMPindex(ip-1)+1, SMPindex(ip)
        W_DNRM20(ip)= W_DNRM20(ip) + WW(i,R)**2
      enddo
!$omp end parallel

      DNRM20= 0.d0
      do ip0= 1, PEsmpTOT
         DNRM20= DNRM20 + W_DNRM20(ip0)
      enddo
      call MPI_Allreduce (DNRM20, DNRM2,…)

      RESID= dsqrt(DNRM2/BNRM2)
      RHO1 = RHO
…
      enddo
```

NOT parallel

Mod.A & B (1/5)

Mod.B: src4

```
!C +---------------+
!C | {z}= [Minv]{r} |
!C +---------------+
!C===
!$omp parallel private(ip,i,ip0)
      ip= omp_get_thread_num() + 1
      do i= SMPindex(ip-1)+1, SMPindex(ip)
        WW(i,Z)= WW(i,R) * WW(i,DD)
      enddo
!C===

!C +--------------+
!C | {RHO}= {r}{z} |
!C +--------------+
!C===
      W_RHO0(ip)= 0.0d0
      do i= SMPindex(ip-1)+1, SMPindex(ip)
        W_RHO0(ip)= W_RHO0(ip) + WW(i,R)*WW(i,Z)
      enddo
!$omp barrier
!$omp master
      RHO0= 0.d0
      do ip0= 1, PEsmpTOT
         RHO0= RHO0 + W_RHO0(ip0)
      enddo
      call MPI_Allreduce (RHO0, RHO, ···)
!$omp end master
!C===
```

Only Master Thread

"!$omp parallel" is still active (fork)

Mod.A: src3

```
!C +---------------+
!C | {z}= [Minv]{r} |
!C +---------------+
!C===
!$omp parallel private(ip,i)
      ip= omp_get_thread_num() + 1
      do i= SMPindex(ip-1)+1, SMPindex(ip)
        WW(i,Z)= WW(i,R) * WW(i,DD)
      enddo
!C===

!C +--------------+
!C | {RHO}= {r}{z} |
!C +--------------+
!C===
      W_RHO0(ip)= 0.0d0
      do i= SMPindex(ip-1)+1, SMPindex(ip)
        W_RHO0(ip)= W_RHO0(ip) + WW(i,R)*WW(i,Z)
      enddo
!$omp end parallel

      RHO0= 0.d0
      do ip0= 1, PEsmpTOT
         RHO0= RHO0 + W_RHO0(ip0)
      enddo
      call MPI_Allreduce (RHO0, RHO, ···)
!C===
```

NOT parallel

Mod.A & B (2/5)

Mod.B: src4

```
!C
!C +-------------------------+
!C | {p} = {z} if      ITER=1    |
!C | BETA= RHO / RHO1  otherwise |
!C +-------------------------+
!C===

!$omp barrier

      if ( ITER.eq.1 ) then
        do i= SMPindex(ip-1)+1, SMPindex(ip)
          WW(i,P)= WW(i,Z)
        enddo
       else
         BETA= RHO / RHO1
        do i= SMPindex(ip-1)+1, SMPindex(ip)
          WW(i,P)= WW(i,Z) + BETA*WW(i,P)
        enddo
      endif
!$omp end parallel
!C===
```

Mod.A: src3

```
!C
!C +-------------------------+
!C | {p} = {z} if      ITER=1    |
!C | BETA= RHO / RHO1  otherwise |
!C +-------------------------+
!C===

!$omp parallel private(ip,i)
      ip= omp_get_thread_num() + 1
      if ( ITER.eq.1 ) then
        do i= SMPindex(ip-1)+1, SMPindex(ip)
          WW(i,P)= WW(i,Z)
        enddo
       else
         BETA= RHO / RHO1
        do i= SMPindex(ip-1)+1, SMPindex(ip)
          WW(i,P)= WW(i,Z) + BETA*WW(i,P)
        enddo
      endif
!$omp end parallel
!C===
```

Mod.A & B (3/5)

Mod.B: src4

```
!C
!C +-------------+
!C | {q}= [A]{p} |
!C +-------------+
!C===
!C
!C-- INTERFACE data EXCHANGE

       call SOLVER_SEND_RECV (…)

!$omp parallel private(ip,ip0,i,j,k,WVAL)
      ip= omp_get_thread_num() + 1

      do j= SMPindex(ip-1)+1, SMPindex(ip)
        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===
```

Mod.A: src3

```
!C
!C +-------------+
!C | {q}= [A]{p} |
!C +-------------+
!C===
!C
!C-- INTERFACE data EXCHANGE

      call SOLVER_SEND_RECV (…)

!$omp parallel private(ip,ip0,i,j,k,WVAL)
      ip= omp_get_thread_num() + 1

      do j= SMPindex(ip-1)+1, SMPindex(ip)
        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===
```

“!$omp parallel” is still active (fork)

Mod.A & B (4/5)

Mod.B: src4

```
!C
!C +-------------------+
!C | ALPHA= RHO / {p}{q} |
!C +-------------------+
!C===
      W_C10(ip)= 0.0d0
      do i= SMPindex(ip-1)+1, SMPindex(ip)
        W_C10(ip)= W_C10(ip) + WW(i,P)*WW(i,Q)
      enddo
!$omp barrier

!$omp master
      C10= 0.d0
      do ip0= 1, PEsmpTOT
         C10= C10 + W_C10(ip0)
      enddo

      call MPI_Allreduce (C10, C1, …)
!$omp end master

!$omp barrier
      ALPHA= RHO / C1
!C===
```

Only Master Thread

“!$omp parallel” is still active (fork)

Mod.A: src3

```
!C
!C +-------------------+
!C | ALPHA= RHO / {p}{q} |
!C +-------------------+
!C===
      W_C10(ip)= 0.0d0
      do i= SMPindex(ip-1)+1, SMPindex(ip)
        W_C10(ip)= W_C10(ip) + WW(i,P)*WW(i,Q)
      enddo
!$omp end parallel


      C10= 0.d0
      do ip0= 1, PEsmpTOT
         C10= C10 + W_C10(ip0)
      enddo

      call MPI_Allreduce (C10, C1, …)



      ALPHA= RHO / C1
!C===
```

NOT parallel

Mod.A & B (5/5)

Mod.B: src4

```
!C
!C +--------------------+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +--------------------+
!C===


      do i= SMPindex(ip-1)+1, SMPindex(ip)
         X(i)  = X (i)   + ALPHA * WW(i,P)
        WW(i,R)= WW(i,R) - ALPHA * WW(i,Q)
      enddo

      W_DNRM20(ip)= 0.0d0
      do i= SMPindex(ip-1)+1, SMPindex(ip)
        W_DNRM20(ip)= W_DNRM20(ip) + WW(i,R)**2
      enddo
!$omp end parallel

      DNRM20= 0.d0
      do ip0= 1, PEsmpTOT
         DNRM20= DNRM20 + W_DNRM20(ip0)
      enddo
      call MPI_Allreduce (DNRM20, DNRM2,…)

      RESID= dsqrt(DNRM2/BNRM2)
      RHO1 = RHO
…
      enddo
```

NOT parallel

Mod.A: src3

```
!C
!C +--------------------+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +--------------------+
!C===
!$omp parallel private(ip,ip0,i)
      ip= omp_get_thread_num() + 1
      do i= SMPindex(ip-1)+1, SMPindex(ip)
         X(i)  = X (i)   + ALPHA * WW(i,P)
        WW(i,R)= WW(i,R) - ALPHA * WW(i,Q)
      enddo

      W_DNRM20(ip)= 0.0d0
      do i= SMPindex(ip-1)+1, SMPindex(ip)
        W_DNRM20(ip)= W_DNRM20(ip) + WW(i,R)**2
      enddo
!$omp end parallel

      DNRM20= 0.d0
      do ip0= 1, PEsmpTOT
         DNRM20= DNRM20 + W_DNRM20(ip0)
      enddo
      call MPI_Allreduce (DNRM20, DNRM2,…)

      RESID= dsqrt(DNRM2/BNRM2)
      RHO1 = RHO
…
      enddo
```

NOT parallel

各実装の特徴

- Original：src1, src2
  - 全ループに *!$omp parallel do/#pragma omp parallel for*
- Mod.A: src3
  - *!$omp parallel/#pragma omp parallel* blocks: 4 blocks
  - NO *!$omp do/#pragma omp for*
- Mod.B: src4
  - *!$omp parallel/#pragma omp parallel* blocks: 2 blocks
    - SEND-RECVの前後
    - 1ブロックにすることも可能
  - NO *!$omp do/#pragma omp for*
  - オーバーヘッド： *!$omp master/#pragma omp master*

OpenMP（Mod.A, Mod.B）（F・C）

```
>$ cd /work/gt00/t00XXX/pFEM/pfem3d/src3
>$ module load fj
>$ make
>$ cd ../run
>$ ls sol3
      sol3

>$ cd /work/gt00/t00XXX/pFEM/pfem3d/src4
>$ module load fj
>$ make
>$ cd ../run
>$ ls sol4
      sol4
```

Mod.A: src3

Mod.B: src4

y12.sh

```
#!/bin/sh
#PJM -N "hb-12"
#PJM -L rscgrp=tutorial-o
#PJM -L node=12
#PJM --mpi proc=48
#PJM --omp thread=12
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o y12.lst

module load fj
module load fjmpi

export OMP_NUM_THREADS=12
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

mpiexec ./sol2
mpiexec ./sol3
mpiexec ./sol4
mpiexec numactl -l ./sol2
mpiexec numactl -l ./sol3
mpiexec numactl -l ./sol4
```

Speed-up of Mat-Ass-Main N=256x256x192, 12-nodes

Time for PCG Solver