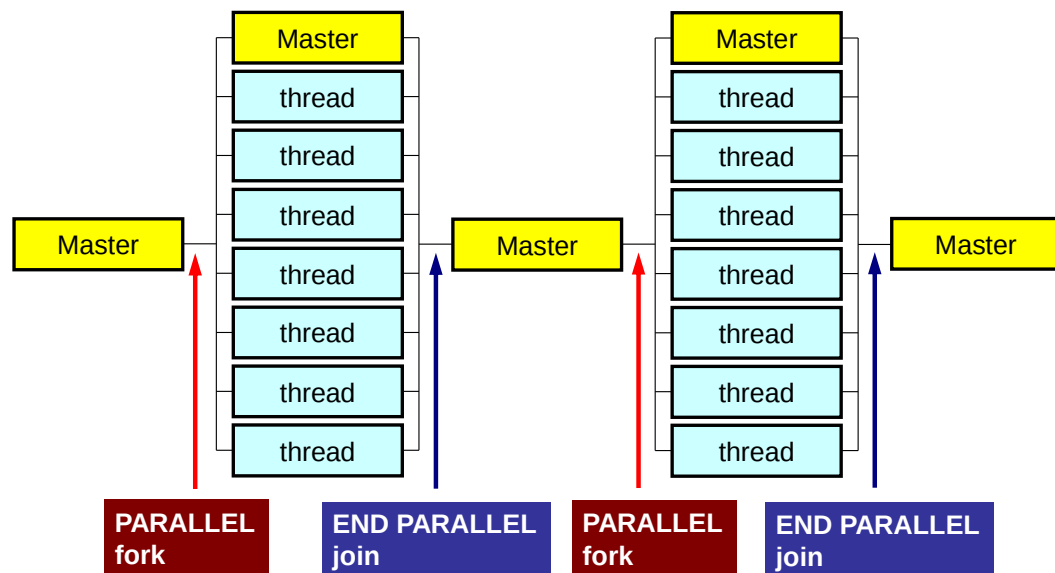


3D Parallel FEM (V) (OpenMP + MPI) Hybrid Parallel Programming Model Further Optimization

Fortran

Kengo Nakajima
RIKEN R-CCS

omp parallel (do)



- “omp parallel-omp end parallel” = “fork-join”
- If you have many loops, these “fork-join’s” cause overheads.
- **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

Mod.A: src3

```
!$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

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

  allocate (SMPindex(0:PEsmpTOT))
  allocate (W_RH00 (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 | {RH0}= {r} {z} |
!C +-----+
!C===

RH00= 0. d0

!$omp parallel do private(i) reduction (+:RH00)
do i= 1, N
    RH00= RH00 + WW(i, R)*WW(i, Z)
enddo

call MPI_Allreduce (RH00, RH0, ...)
!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 | {RH0}= {r} {z} |
!C +-----+
!C===

W_RH00(ip)= 0. 0d0
do i= SMPindex(ip-1)+1, SMPindex(ip)
    W_RH00(ip)= W_RH00(ip) + WW(i, R)*WW(i, Z)
enddo
!$omp end parallel

RH00= 0. d0
do ip0= 1, PEsmpTOT
    RH00= RH00 + W_RH00(ip0)
enddo
call MPI_Allreduce (RH00, RH0, ...)
!C===

```

NOT parallel

SOLVER_CG (2/5)

Original: src2

```
!C
!C +-----+
!C | {p} = {z} if      ITER=1 |
!C | BETA= RHO / RH01 otherwise |
!C +-----+
!C==
      if ( ITER. eq. 1 ) then

!$omp parallel do private(i)
      do i= 1, N
        WW(i,P)= WW(i,Z)
      enddo
      else
        BETA= RHO / RH01
!$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 / RH01 otherwise |
!C +-----+
!C==

!$omp parallel private(ip,i)
      ip= omp_get_thread_num() + 1
      if ( ITER. eq. 1 ) then
        do i= SMPindex(ip-1)+1, SMPindex(ip)
          WW(i,P)= WW(i,Z)
        enddo
      else
        BETA= RHO / RH01
        do i= SMPindex(ip-1)+1, SMPindex(ip)
          WW(i,P)= WW(i,Z) + BETA*WW(i,P)
        enddo
      endif
!$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)

RH01 = RH0

...

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)

RH01 = RH0

...

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 | {RH0}= {r} {z} |
!C +-----+
!C===
    W_RH00(ip)= 0.0d0
    do i= SMPindex(ip-1)+1, SMPindex(ip)
        W_RH00(ip)= W_RH00(ip) + WW(i,R)*WW(i,Z)
    enddo
!$omp barrier
!$omp master
    RH00= 0.d0
    do ip0= 1, PEsmptOT
        RH00= RH00 + W_RH00(ip0)
    enddo
    call MPI_Allreduce (RH00, RH0, ...)
!$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 | {RH0}= {r} {z} |
!C +-----+
!C===
    W_RH00(ip)= 0.0d0
    do i= SMPindex(ip-1)+1, SMPindex(ip)
        W_RH00(ip)= W_RH00(ip) + WW(i,R)*WW(i,Z)
    enddo
!$omp end parallel

    RH00= 0.d0
    do ip0= 1, PEsmptOT
        RH00= RH00 + W_RH00(ip0)
    enddo
    call MPI_Allreduce (RH00, RH0, ...)
!C===

```

NOT parallel

Mod.A & B (2/5)

Mod.B: src4

```
!C
!C +-----+
!C | {p} = {z} if      ITER=1 |
!C | BETA= RHO / RH01 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 / RH01
  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 / RH01 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 / RH01
  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)
RH01 = RH0
```

```
...
```

```
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)
RH01 = RH0
```

```
...
```

```
enddo
```

NOT parallel

Features of Each Implementation

- Original
 - All loops are *!\$omp parallel do/#pragma omp parallel for*
- Mod.A
 - *!\$omp parallel/#pragma omp parallel* blocks: 4 blocks
 - NO *!\$omp do/#pragma omp for*
- Mod.B
 - *!\$omp parallel/#pragma omp parallel* blocks: 2 blocks
 - Before/After SEND-RECV
 - Could be a Single Block
 - NO *!\$omp do/#pragma omp for*
 - Overhead of *!\$omp master/#pragma omp master*
 - Overhead of *!\$omp barrier/#pragma omp barrier*
- Mod.A and Mod.B are better if thread# is larger

OpenMP (Mod.A, Mod.B) (F·C)

```
>$ cd /work/gt36/t36XXX/pFEM/pfem3d/src3
```

Mod.A: src3

```
>$ module load fj
```

```
>$ make
```

```
>$ cd ../run
```

```
>$ ls sol3  
sol3
```

```
>$ cd /work/gt36/t36XXX/pFEM/pfem3d/src4
```

Mod.B: src4

```
>$ module load fj
```

```
>$ make
```

```
>$ cd ../run
```

```
>$ ls sol4  
sol4
```

y12.sh

```
#!/bin/sh
#PJM -N "hb-12"
#PJM -L rscgrp=lecture5-o
#PJM -L node=12
#PJM -mpi proc=48
#PJM -omp thread=12
#PJM -L elapse=00:15:00
#PJM -g gt55
#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
```

Time for PCG Solver

$N=256 \times 256 \times 192$, 12-nodes

