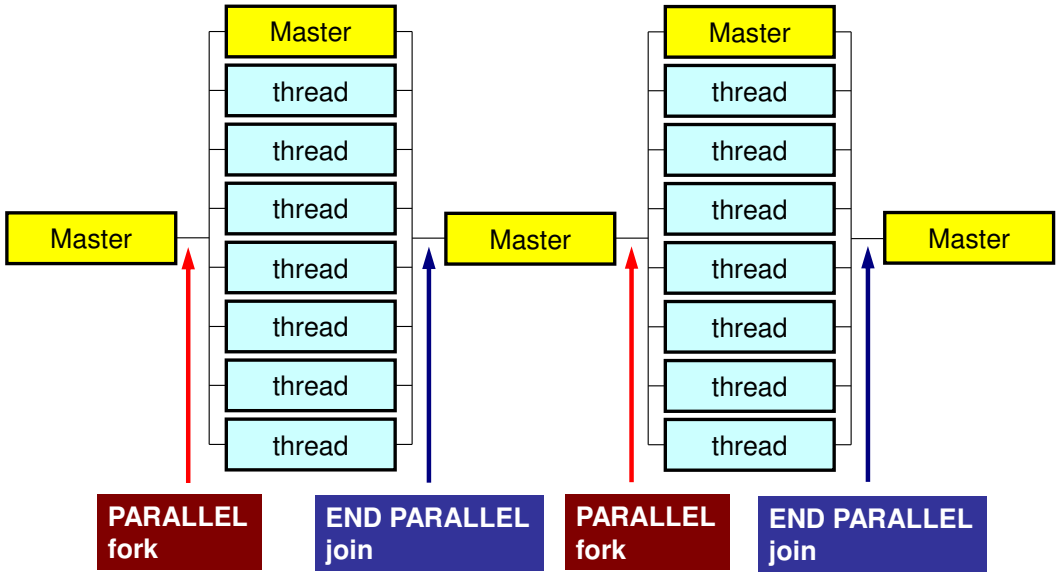
3D Parallel FEM (V)
(OpenMP + MPI) Hybrid Parallel Programming Model
Further Optimization

C Language

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 operations
- 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

```
#pragma omp parallel for{
for(i=0:i<N;i++){
(…)
} }
```

```
#pragma omp parallel{

ip=omp_get_thread_num();
for(i=SMPindex[ip];
    i<SMPindex[ip+1];
    i++){
(…)
}
}
```

PEsmpTOT: Total Number of Threads

```
#pragma omp parallel
{
#pragma omp master
  {PEsmpTOT= omp_get_num_threads();}
}

ERROR= 0;

COMPtime=0.0;
COMMtime=0.0;

WW=(KREAL**) allocate_matrix(sizeof(KREAL),4,NP);

allocate_vector (KREAL)WS[NP], WR[NP]
allocate_vector (KREAL)W_RHO0[PEsmpTOT], W_C10[PEsmpTOT],
                       W_BNRM20[PEsmpTOT],
                       W_DNRM20[PEsmpTOT]
allocate_vector (KINT) SMPindex[PEsmpTOT+1]

nth= N/PEsmpTOT; nr = N - nth*PEsmpTOT;

for(ip=1;ip<PEsmpTOT+1;ip++) {
  SMPindex[ip]= nth;
  if (ip <= nr) SMPindex[ip]=nth+1;
  }
  for(ip=1;ip<PEsmpTOT+1;ip++) {
    SMPindex[ip]= SMPindex[ip-1] + SMPindex[ip];
  }
```

SOLVER_CG (1/5)

Original: src2

```
/**
   +---------------+
   | {z}= [Minv]{r} |
   +---------------+
**/
#pragma omp parallel for private (i)
    for(i=0;i<N;i++){
      WW[Z][i]= WW[DD][i]*WW[R][i];
    }
/**
   +---------------+
   | {RHO}= {r}{z} |
   +---------------+
**/
    RHO0= 0.e0;

#pragma omp parallel for private (i) reduction (+:RHO0)
    for(i=0;i<N;i++){
      RHO0+= WW[R][i]*WW[Z][i];
    }

    MPI_Allreduce (&RHO0, &RHO,…);
```

Mod.A: src3

```
/**
   +---------------+
   | {z}= [Minv]{r} |
   +---------------+
**/
#pragma omp parallel private (ip,i,RHO0)
  {
    ip= omp_get_thread_num();
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      WW[Z][i]= WW[DD][i]*WW[R][i];
    }
/**
   +---------------+
   | {RHO}= {r}{z} |
   +---------------+
**/
    W_RHO0[ip]=0.0;
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      W_RHO0[ip]+= WW[R][i]*WW[Z][i];
    }
  }
/** END PARALLEL **/

    RHO0= 0.0;
    for(ip0=0;ip0<PEsmpTOT;ip0++){
      RHO0+= W_RHO0[ip0];
    }

    MPI_Allreduce (&RHO0, &RHO,…);
```

NOT parallel

SOLVER_CG (2/5)

Original: src2

```
/**
   +----------------------------+
   | {p} = {z} if      ITER=1   |
   | BETA= RHO / RHO1  otherwise |
   +----------------------------+
**/
    if( ITER == 1 ){
#pragma omp parallel for private (i)
      for(i=0;i<N;i++){
        WW[P][i]=WW[Z][i];
      }
    }else{
      BETA= RHO / RHO1;
#pragma omp parallel for private (i)
      for(i=0;i<N;i++){
        WW[P][i]=WW[Z][i] + BETA*WW[P][i];
      }
    }
```

Mod.A: src3

```
**
   +----------------------------+
   | {p} = {z} if      ITER=1   |
   | BETA= RHO / RHO1  otherwise |
   +----------------------------+
**/
#pragma omp parallel private (ip,i)
  {
    ip= omp_get_thread_num();
    if( ITER == 1 ){
      for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
        WW[P][i]=WW[Z][i];
      }
    }else{
      BETA= RHO / RHO1;
      for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
        WW[P][i]=WW[Z][i] + BETA*WW[P][i];
      }
    }
  }
/** END PARALLEL **/
```

SOLVER_CG (3/5)

Original: src2

```
/**
   +-------------+
   | {q}= [A]{p} |
   +-------------+
**/

    SOLVER_SEND_RECV (…);

#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;
    }
```

Mod.A: src3

```
/**
   +-------------+
   | {q}= [A]{p} |
   +-------------+
**/

    SOLVER_SEND_RECV(…);

#pragma omp parallel private (ip,j,i,k,WVAL)
  {
    ip= omp_get_thread_num();
    for(j=SMPindex[ip];j<SMPindex[ip+1];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;
    }
```

“#pragma omp parallel” is still active (fork)

SOLVER_CG (4/5)

Original: src2

```
/**
   +--------------------+
   | ALPHA= RHO / {p}{q} |
   +--------------------+
**/
    C10= 0.e0;
#pragma omp parallel for private (i) reduction (+:C10)
    for(i=0;i<N;i++){
      C10+=WW[P][i]*WW[Q][i];
    }
    MPI_Allreduce (&C10,&C1,…);

    ALPHA= RHO / C1;
```

Mod.A: src3

```
/**
   +--------------------+
   | ALPHA= RHO / {p}{q} |
   +--------------------+
**/
    W_C10[ip]=0.0;
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      W_C10[ip]+= WW[P][i]*WW[Q][i];
    }
  }
/** END PARALLEL **/

    C10= 0.0;
    for(ip0=0;ip0<PEsmpTOT;ip0++){
      C10+= W_C10[ip0];
    }

    MPI_Allreduce (&C10,&C1,…);
    ALPHA= RHO / C1;
```

NOT parallel

SOLVER_CG (5/5)

Original: src2

```
**
   +---------------------+
   | {x}= {x} + ALPHA*{p} |
   | {r}= {r} - ALPHA*{q} |
   +---------------------+
**/
#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];
    }

    MPI_Allreduce (&DNRM20,&DNRM2, ...);
    RESID= sqrt(DNRM2/BNRM2);
```

Mod.A: src3

```
/**
   +---------------------+
   | {x}= {x} + ALPHA*{p} |
   | {r}= {r} - ALPHA*{q} |
   +---------------------+
**/
#pragma omp parallel private (ip,i)
  {
    ip= omp_get_thread_num();
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      X [i]    +=  ALPHA *WW[P][i];
      WW[R][i]+= -ALPHA *WW[Q][i];
    }

    W_DNRM20[ip]=0.0;
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      W_DNRM20[ip]+= WW[R][i]*WW[R][i];
    }
  }
/** END PARALLEL **/

    DNRM20= 0.e0;
    for(ip0=0;ip0<PEsmpTOT;ip0++){
      DNRM20+= W_DNRM20[ip0];
    }

    MPI_Allreduce (&DNRM20,&DNRM2,...);
    RESID= sqrt(DNRM2/BNRM2);
```

NOT parallel

Mod.A & B (1/5)

Mod.B: src4

```
/**
   +---------------+
   | {z}= [Minv]{r} |
   +---------------+
**/
#pragma omp parallel private (ip,i,RHO0)
  {
    ip= omp_get_thread_num();
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      WW[Z][i]= WW[DD][i]*WW[R][i];
    }
/**
   +--------------+
   | {RHO}= {r}{z} |
   +--------------+
**/
    W_RHO0[ip]=0.0;
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      W_RHO0[ip]+= WW[R][i]*WW[Z][i];
    }
#pragma omp barrier

#pragma omp master
  {
    RHO0= 0.0;
    for(ip0=0;ip0<PEsmpTOT;ip0++){
      RHO0+= W_RHO0[ip0];
    }

    MPI_Allreduce (&RHO0, &RHO,…);
}
```

Only Master Thread

“#pragma omp parallel” is still active (fork)

Mod.A: src3

```
/**
   +---------------+
   | {z}= [Minv]{r} |
   +---------------+
**/
#pragma omp parallel private (ip,i,RHO0)
  {
    ip= omp_get_thread_num();
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      WW[Z][i]= WW[DD][i]*WW[R][i];
    }
/**
   +--------------+
   | {RHO}= {r}{z} |
   +--------------+
**/
    W_RHO0[ip]=0.0;
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      W_RHO0[ip]+= WW[R][i]*WW[Z][i];
    }
  }
/** END PARALLEL **/

    RHO0= 0.0;
    for(ip0=0;ip0<PEsmpTOT;ip0++){
      RHO0+= W_RHO0[ip0];
    }

    MPI_Allreduce (&RHO0, &RHO,…);
```

NOT parallel

Mod.A & B (2/5)

Mod.B: src4

```
/**
   +----------------------------+
   | {p} = {z} if       ITER=1  |
   | BETA= RHO / RHO1   otherwise |
   +----------------------------+
**/
#pragma omp barrier
    if( ITER == 1 ){
      for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
        WW[P][i]=WW[Z][i];
      }
    }else{
      BETA= RHO / RHO1;
      for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
        WW[P][i]=WW[Z][i] + BETA*WW[P][i];
      }
    }
  }
/** END PARALLEL **/
```

Mod.A: src3

```
**
   +----------------------------+
   | {p} = {z} if       ITER=1  |
   | BETA= RHO / RHO1   otherwise |
   +----------------------------+
**/
#pragma omp parallel private (ip,i)
  {
    ip= omp_get_thread_num();
    if( ITER == 1 ){
      for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
        WW[P][i]=WW[Z][i];
      }
    }else{
      BETA= RHO / RHO1;
      for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
        WW[P][i]=WW[Z][i] + BETA*WW[P][i];
      }
    }
  }
/** END PARALLEL **/
```

Mod.A & B (3/5)

Mod.B: src4

```
/**
   +-------------+
   | {q}= [A]{p} |
   +-------------+
**/

    SOLVER_SEND_RECV(…);

#pragma omp parallel private (ip,j,i,k,WVAL)
  {
    ip= omp_get_thread_num();
    for(j=SMPindex[ip];j<SMPindex[ip+1];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;
    }
```

Mod.A: src3

```
/**
   +-------------+
   | {q}= [A]{p} |
   +-------------+
**/

    SOLVER_SEND_RECV(…);

#pragma omp parallel private (ip,j,i,k,WVAL)
  {
    ip= omp_get_thread_num();
    for(j=SMPindex[ip];j<SMPindex[ip+1];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;
    }
```

“#pragma omp parallel” is still active (fork)

Mod.A & B (4/5)

Mod.B: src4

```
/**
   +--------------------+
   | ALPHA= RHO / {p}{q} |
   +--------------------+
**/
    W_C10[ip]=0.0;
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      W_C10[ip]+= WW[P][i]*WW[Q][i];
    }
#pragma omp barrier

#pragma omp master
    {
    C10= 0.0;
    for(ip0=0;ip0<PEsmpTOT;ip0++){
      C10+= W_C10[ip0];
    }

    MPI_Allreduce (&C10,&C1,…);
    }

#pragma omp barrier
    ALPHA= RHO / C1;
```

Only Master Thread

“#pragma omp parallel” is still active (fork)

Mod.A: src3

```
/**
   +--------------------+
   | ALPHA= RHO / {p}{q} |
   +--------------------+
**/
    W_C10[ip]=0.0;
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      W_C10[ip]+= WW[P][i]*WW[Q][i];
    }
  }
/** END PARALLEL **/

    C10= 0.0;
    for(ip0=0;ip0<PEsmpTOT;ip0++){
      C10+= W_C10[ip0];
    }

    MPI_Allreduce (&C10,&C1,…);
    ALPHA= RHO / C1;
```

NOT parallel

Mod.A & B (5/5)

Mod.B: src4

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



    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      X [i]    +=  ALPHA *WW[P][i];
      WW[R][i]+= -ALPHA *WW[Q][i];
    }

    W_DNRM20[ip]=0.0;
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      W_DNRM20[ip]+= WW[R][i]*WW[R][i];
    }
  }
/** END PARALLEL **/

    DNRM20= 0.e0;
    for(ip0=0;ip0<PEsmpTOT;ip0++){
      DNRM20+= W_DNRM20[ip0];
    }

    MPI_Allreduce (&DNRM20,&DNRM2,…);
    RESID= sqrt(DNRM2/BNRM2);
```

NOT parallel

Mod.A: src3

```
/**
   +---------------------+
   | {x}= {x} + ALPHA*{p} |
   | {r}= {r} - ALPHA*{q} |
   +---------------------+
**/
#pragma omp parallel private (ip,i)
  {
    ip= omp_get_thread_num();
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      X [i]    +=  ALPHA *WW[P][i];
      WW[R][i]+= -ALPHA *WW[Q][i];
    }

    W_DNRM20[ip]=0.0;
    for(i=SMPindex[ip];i<SMPindex[ip+1];i++){
      W_DNRM20[ip]+= WW[R][i]*WW[R][i];
    }
  }
/** END PARALLEL **/

    DNRM20= 0.e0;
    for(ip0=0;ip0<PEsmpTOT;ip0++){
      DNRM20+= W_DNRM20[ip0];
    }

    MPI_Allreduce (&DNRM20,&DNRM2,…);
    RESID= sqrt(DNRM2/BNRM2);
```

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*
- Mod.A and Mod.B are better if thread# is larger

OpenMP（Mod.A, Mod.B）（F・C）

```
>$ cd /home/ra020019/<Your-UID>/pFEM/pfem3d/src3
>$ make
>$ cd ../run
>$ ls sol3
     sol3

>$ cd /home/ra020019/<Your-UID>/pFEM/pfem3d/src4
>$ make
>$ cd ../run
>$ ls sol4
     sol4
```

Mod.A: src3

Mod.B: src4

y12.sh

```
#!/bin/bash
#PJM -N "hb-12"
#PJM -L “rscgrp=small”
#PJM -L “node=12”
#PJM --mpi “max-proc-per-node=4”
#PJM -L elapse=00:15:00
#PJM -g ra020019
#PJM -j
#PJM -e err
#PJM -o y12.lst


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=256x256x192, 12-nodes