3D Parallel FEM (IV)
OpenMP + Hybrid Parallel Programming Model

Kengo Nakajima
Information Technology Center

Programming for Parallel Computing (616-2057)
Seminar on Advanced Computing (616-4009)
Hybrid Parallel Programming Model

- Message Passing (e.g. MPI) + Multi Threading (e.g. OpenMP, CUDA, OpenCL, OpenACC etc.)
- In K computer and FX10, hybrid parallel programming is recommended
  - MPI + Automatic Parallelization by Fujitsu’s Compiler
  - Personally, I do not like to call this “hybrid” !!!
- Expectations for Hybrid
  - Number of MPI processes (and sub-domains) to be reduced
  - $O(10^8-10^9)$-way MPI might not scale in Exascale Systems
  - Easily extended to Heterogeneous Architectures
    - CPU+GPU, CPU+Manycores (e.g. Intel MIC/Xeon Phi)
    - MPI+X: OpenMP, OpenACC, CUDA, OpenCL
Flat MPI vs. Hybrid

Flat-MPI: Each Core -> Independent

Hybrid: Hierarchical Structure
HB  M  x  N

Number of OpenMP threads per a single MPI process

Number of MPI process per a single node
Size (and number) of local data changes according to parallel programming model

example: 6 nodes, 96 cores

Flat MPI

- 128 192 64
- 8 12 1
- pcube

HB 4x4

- 128 192 64
- 4 6 1
- pcube

HB 16x1

- 128 192 64
- 2 3 1
- pcube
Batch Script (1/2)
Env. Var.: OMP_NUM_THREADS

Flat MPI

```bash
#!/bin/sh

#PJM -L "node=6"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture2"
#PJM -g "gt82"
#PJM -o "test.lst"
#PJM --mpi "proc=96"

mpiexec ./sol

rm wk.*
```

Hybrid 16 × 1

```bash
#!/bin/sh

#PJM -L "node=6"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture2"
#PJM -g "gt82"
#PJM -o "test.lst"
#PJM --mpi "proc=6"

export OMP_NUM_THREADS=16
mpiexec ./sol

rm wk.*
```
### Batch Script (2/2)

**Env. Var.: OMP_NUM_THREADS**

<table>
<thead>
<tr>
<th>Hybrid 4 × 4</th>
<th>Hybrid 8 × 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#!/bin/sh</code></td>
<td><code>#!/bin/sh</code></td>
</tr>
<tr>
<td><code>#PJM -L &quot;node=6&quot;</code></td>
<td><code>#PJM -L &quot;node=6&quot;</code></td>
</tr>
<tr>
<td><code>#PJM -L &quot;elapse=00:05:00&quot;</code></td>
<td><code>#PJM -L &quot;elapse=00:05:00&quot;</code></td>
</tr>
<tr>
<td><code>#PJM -j</code></td>
<td><code>#PJM -j</code></td>
</tr>
<tr>
<td><code>#PJM -L &quot;rscgrp=lecture2&quot;</code></td>
<td><code>#PJM -L &quot;rscgrp=lecture2&quot;</code></td>
</tr>
<tr>
<td><code>#PJM -g &quot;gt82&quot;</code></td>
<td><code>#PJM -g &quot;gt82&quot;</code></td>
</tr>
<tr>
<td><code>#PJM -o &quot;test.lst&quot;</code></td>
<td><code>#PJM -o &quot;test.lst&quot;</code></td>
</tr>
<tr>
<td><code>#PJM --mpi &quot;proc=24&quot;</code></td>
<td><code>#PJM --mpi &quot;proc=12&quot;</code></td>
</tr>
<tr>
<td><code>export OMP_NUM_THREADS=4</code></td>
<td><code>export OMP_NUM_THREADS=8</code></td>
</tr>
<tr>
<td><code>mpiexec ./sol</code></td>
<td><code>mpiexec ./sol</code></td>
</tr>
<tr>
<td><code>rm wk.*</code></td>
<td><code>rm wk.*</code></td>
</tr>
</tbody>
</table>
Background

• Multicore/Manycore Processors
  – Low power consumption, Various types of programming models

• OpenMP
  – Directive based, (seems to be) easy
  – Many books

• Data Dependency (no classes this year)
  – Conflict of reading from/writing to memory
  – Appropriate reordering of data is needed for “consistent” parallel computing
  – NO detailed information in OpenMP books: very complicated

• OpenMP/MPI Hybrid Parallel Programming Model for Multicore/Manycore Clusters
• SMP
  – Symmetric Multi Processors
  – Multiple CPU’s (cores) share a single memory space
What is OpenMP?

http://www.openmp.org

- An API for multi-platform shared-memory parallel programming in C/C++ and Fortran
  - Current version: 4.0
- Background
  - Merger of Cray and SGI in 1996
  - ASCI project (DOE) started
- C/C++ version and Fortran version have been separately developed until ver.2.5.
- Fork-Join Parallel Execution Model
- Users have to specify everything by directives.
  - Nothing happen, if there are no directives
Fork-Join Parallel Execution Model
Number of Threads

• **OMP_NUM_THREADS**
  - How to change?
    • bash (.bashrc)  
      `export OMP_NUM_THREADS=8`
    • csh (.cshrc)  
      `setenv OMP_NUM_THREADS 8`
Information about OpenMP

• OpenMP Architecture Review Board (ARB)
  – http://www.openmp.org

• References
  – 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）最新！

• Japanese Version of OpenMP 3.0 Spec. (Fujitsu etc.)
Features of OpenMP

• Directives
  – Loops right after the directives are parallelized.
  – If the compiler does not support OpenMP, directives are considered as just comments.
OpenMP/Directives
Array Operations

Simple Substitution

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

Dot Products

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

DAXPY

```c
!$omp parallel do
do i = 1, NP
    Y(i) = ALPHA*X(i) + Y(i)
endo
!$omp end parallel do
```
OpenMP/Directives
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
Features of OpenMP

• Directives
  – Loops right after the directives are parallelized.
  – If the compiler does not support OpenMP, directives are considered as just comments.

• Nothing happen without explicit directives
  – Different from “automatic parallelization/vectorization”
  – Something wrong may happen by un-proper way of usage
  – Data configuration, ordering etc. are done under users’ responsibility

• “Threads” are created according to the number of cores on the node
  – Thread: “Process” in MPI
  – Generally, “# threads = # cores”: Xeon Phi supports 4 threads per core (Hyper Multithreading)
Memory Contention: メモリ競合

- During a complicated process, multiple threads may simultaneously try to update the data in same address on the memory.
  - e.g.: Multiple cores update a single component of an array.
  - This situation is possible.
  - Answers may change compared to serial cases with a single core (thread).
Memory Contention (cont.)

• In this lecture, no such case does not happen by reordering etc.
  – In OpenMP, users are responsible for such issues (e.g. proper data configuration, reordering etc.)

• Generally speaking, performance per core reduces as number of used cores (thread number) increases.
  – Memory access performance: STREAM
Features of OpenMP (cont.)

- **"!omp parallel do"-"!omp end parallel do"**
- **Global (Shared) Variables, Private Variables**
  - Default: Global (Shared)
  - Dot Products: reduction

```fortran
!$omp parallel do private(iS,iE,i)
!$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, PE_{smpTOT} global (shared)
Fortran & C

```fortran
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
end subroutine
```
In this class ...

• There are many capabilities of OpenMP.
• In this class, only several functions are shown for parallelization of parallel FEM.
First things to be done (after OpenMP 3.0)

- use omp_lib Fortran
- #include <omp.h> C
OpenMP Directives (Fortran)

sentinel directive_name [clause[,[,] clause]...]

• NO distinctions between upper and lower cases.
• sentinel
  – Fortran: !$OMP, C$OMP, *$OMP
    • !$OMP only for free format
  – Continuation Lines (Same rule as that of Fortran compiler is applied)
    • Example for !$OMP PARALLEL DO SHARED(A,B,C)

!$OMP PARALLEL DO
!$OMP+SHARED (A,B,C)  !$OMP PARALLEL DO &
!$OMP SHARED (A,B,C)
OpenMP Directives (C)

```c
#pragma omp directive_name [clause[], clause]...
```

- `\`"\`"` for continuation lines
- Only lower case (except names of variables)

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

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

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

- Parallerize DO/for Loops
- Examples of “clause”
  - PRIVATE(list)
  - SHARED(list)
  - DEFAULT(PRIVATE|SHARED|NONE)
  - REDUCTION({operation|intrinsic}: list)
REDUCTION

REDUCTION ({operator|instinsic}: list)

reduction ({operator|instinsic}: list)

- Similar to “MPI_Reduce”
- Operator
  - +, *, -, .AND., .OR., .EQV., .NEQV.
- Intrinsic
  - MAX, MIN, IAND, IOR, IEQR
Example-1: A Simple Loop

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

- Default status of loop variables (“i” in this case) is private. Therefore, explicit declaration is not needed.
- “END PARALLEL DO” is not required
  - In C, there are no definitions of “end parallel do”
Example-1: REDUCTION

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

- "END PARALLEL DO" is not required.
Functions which can be used with OpenMP

<table>
<thead>
<tr>
<th>Name</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>int omp_get_num_threads (void)</td>
<td>Total Thread #</td>
</tr>
<tr>
<td>int omp_get_thread_num (void)</td>
<td>Thread ID</td>
</tr>
<tr>
<td>double omp_get_wtime (void)</td>
<td>= MPI_Wtime</td>
</tr>
<tr>
<td>void omp_set_num_threads (int num_threads)</td>
<td>Setting Thread #</td>
</tr>
<tr>
<td>call omp_set_num_threads (num_threads)</td>
<td></td>
</tr>
</tbody>
</table>
OpenMP for Dot Products

```
VAL= 0. d0
do i= 1, N
   VAL= VAL + W(i, R) * W(i, Z)
endo
```
OpenMP for Dot Products

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

Directives are just inserted.

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

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

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

Directives are just inserted.

Multiple Loop

PEsmpTOT: Number of threads

Additional array INDEX (: ) is needed.
Efficiency is not necessarily good, but users can specify thread for each component of data.
OpenMP for Dot Products

```
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
```

Multiple Loop

PEsmpTOT: Number of threads

Additional array INDEX(:) is needed.
Efficiency is not necessarily good, but users can specify thread for each component of data.

e.g.: N=100, PEsmpTOT=4

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

NOT good for GPU’s
Files on Oakleaf-FX

```bash
> $ cd <$O-TOP>
> $ cp /home/z30088/class_eps/F/omp.tar .
> $ cp /home/z30088/class_eps/C/omp.tar .
> $ tar xvf omp.tar
> $ cd omp
<$O-omp>
```
Files on Oakleaf-FX

```bash
$ cd <$O-omp>

$ frtpx -Kfast,openmp test.f
$ fccpx -Kfast,openmp test.c

$ pjsub go.sh
```
Running the Job

go.sh

#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:10:00"
#PJM -L "rscgrp=lecture"
#PJM -g "gt71"
#PJM -j
#PJM -o "t0-08.lst"

export OMP_NUM_THREADS=8 Number of Threads/process

./a.out < INPUT.DAT

INPUT.DAT

N nopt

N: Problem Size (Vector Length)
nopt: First-touch (0:No, 1:Yes)
• DAXPY
• Dot Products
• Effect of OpenMP Directives
• Effect of First Touch Data Placement
use omp_lib
implicit REAL*8 (A-H,O-Z)
real(kind=8), dimension(:,), allocatable :: X, Y
real(kind=8), dimension(:,), allocatable :: Z1, Z2
real(kind=8), dimension(:,), allocatable :: Z3, Z4, Z5
integer, dimension(0:2) :: INDEX

!C +------!
!C | INIT |
!C +------!
!C===
write (*,*) 'N, nopt ?'
read  (*,*)  N, nopt
allocate (X(N), Y(N), Z1(N), Z2(N), Z3(N), Z4(N), Z5(N))
if (nopt.eq.0) then
  X = 1.d0
  Y = 1.d0
  Z1= 0.d0
  Z2= 0.d0
  Z3= 0.d0
  Z4= 0.d0
  Z5= 0.d0
else
!$omp parallel do private (i)
  do i= 1, N
    X (i)= 0.d0
    Y (i)= 0.d0
    Z1(i)= 0.d0
    Z2(i)= 0.d0
    Z3(i)= 0.d0
    Z4(i)= 0.d0
    Z5(i)= 0.d0
  enddo
!$omp end parallel do
endif
ALPHA= 1.d0
!C===

Problem Size, Option

nopt=0  NO First Touch
nopt≠0  with First Touch
test.f (1/3): Initialization

use omp_lib
implicit REAL*8 (A-H,O-Z)
real(kind=8), dimension(:), allocatable :: X, Y
real(kind=8), dimension(:), allocatable :: Z1, Z2
real(kind=8), dimension(:), allocatable :: Z3, Z4, Z5
integer, dimension(0:2) :: INDEX

!C +------+
!C | INIT |
!C +------+
!C ===

write (*,*) 'N, nopt ?'
read (*,*)  N, nopt
allocate (X(N), Y(N), Z1(N), Z2(N), Z3(N), Z4(N), Z5(N))
if (nopt .eq. 0) then
  X = 1.d0
  Y = 1.d0
  Z1= 0.d0
  Z2= 0.d0
  Z3= 0.d0
  Z4= 0.d0
  Z5= 0.d0
else
  !$omp parallel do private (i)
    do i = 1, N
        X (i)= 0.d0
        Y (i)= 0.d0
        Z1(i)= 0.d0
        Z2(i)= 0.d0
        Z3(i)= 0.d0
        Z4(i)= 0.d0
        Z5(i)= 0.d0
    enddo
  !$omp end parallel do
endif
ALPHA= 1.d0

nopt=0
NO First Touch

Initialization is not parallelized
use omp_lib
implicit REAL*8 (A-H,O-Z)
real(kind=8), dimension(:,), allocatable :: X, Y
real(kind=8), dimension(:,), allocatable :: Z1, Z2
real(kind=8), dimension(:,), allocatable :: Z3, Z4, Z5
integer, dimension(0:2) :: INDEX

!C
!C +------+
!C | INIT |
!C +------+
!C===
write (*,*) 'N, nopt ?'
read  (*,*)  N, nopt
allocate (X(N), Y(N), Z1(N), Z2(N), Z3(N), Z4(N), Z5(N))
if (nopt.eq.0) then
  X = 1.d0
  Y = 1.d0
  Z1= 0.d0
  Z2= 0.d0
  Z3= 0.d0
  Z4= 0.d0
  Z5= 0.d0
else
  !$omp parallel do private (i)
  do i= 1, N
    X (i)= 0.d0
    Y (i)= 0.d0
    Z1(i)= 0.d0
    Z2(i)= 0.d0
    Z3(i)= 0.d0
    Z4(i)= 0.d0
    Z5(i)= 0.d0
  enddo
  !$omp end parallel do
endif
ALPHA= 1.d0
!C===

nopt≠0
with First Touch

Initialization is in parallel manner.
test.f (2/3): DAXPY

!C  
!C +-------+  
!C | DAXPY |  
!C +-------+  
!C===
S2time = omp_get_wtime()
$omp parallel do private (i)
    do i = 1, N
        Z1(i) = ALPHA*X(i) + Y(i)
        Z2(i) = ALPHA*X(i) + Y(i)
        Z3(i) = ALPHA*X(i) + Y(i)
        Z4(i) = ALPHA*X(i) + Y(i)
        Z5(i) = ALPHA*X(i) + Y(i)
    enddo
$omp end parallel do
E2time = omp_get_wtime()

write (*, '(a)') '# DAXPY'
write (*, '( a, 1pe16.6)') 'omp-1 ', E2time - S2time
!C===
test.f (3/3): Dot Products

```fortran
!C +-----+!C | DOT |!C +-----+!C===
V1= 0.d0 V2= 0.d0 V3= 0.d0 V4= 0.d0 V5= 0.d0
S2time= omp_get_wtime()
$omp parallel do private(i) reduction (+:V1,V2,V3,V4,V5)
do i= 1, N
  V1= V1 + X(i)*(Y(i)+1.d0)
  V2= V2 + X(i)*(Y(i)+2.d0)
  V3= V3 + X(i)*(Y(i)+3.d0)
  V4= V4 + X(i)*(Y(i)+4.d0)
  V5= V5 + X(i)*(Y(i)+5.d0)
enddo
$omp end parallel do
E2time= omp_get_wtime()
write (*,'(/a)')          '# DOT'
write (*,'( a, 1pe16.6)') 'omp-1 ', E2time - S2time
!C===
stop
end
```
test.c: Dummy Pragma needed

```c
#pragma omp parallel
{

    if (nopt==0) {for (i=0; i<N; i++) {
        X[i] = 1.0;
        Y[i] = 1.0;
        Z1[i] = 0.0;
        Z2[i] = 0.0;
        Z3[i] = 0.0;
        Z4[i] = 0.0;
        Z5[i] = 0.0;
    }
}
else{
    #pragma omp parallel for private(i)
    for (i=0; i<N; i++) {
        X[i] = 1.0;
        Y[i] = 1.0;
        Z1[i] = 0.0;
        Z2[i] = 0.0;
        Z3[i] = 0.0;
        Z4[i] = 0.0;
        Z5[i] = 0.0;
    }
}
```

Dummy Pragma needed before actual computation (for Fujitsu C compiler). This problem may be fixed. Threads for OpenMP are generated during the first parallel loop, and this procedure is expensive. In Fortran, this dummy loop is implicitly inserted by compiler.
DAXPY: Effect of First Touch

- T2K: Effect of first touch is large
- NOT scalable: memory contention, synchronization overhead
First Touch Data Placement

“Patterns for Parallel Programming” Mattson, T.G. et al.

To reduce memory traffic in the system, it is important to keep the data close to the PEs that will work with the data (e.g. NUMA control).

On NUMA computers, this corresponds to making sure the pages of memory are allocated and “owned” by the PEs that will be working with the data contained in the page.

The most common NUMA page-placement algorithm is the “first touch” algorithm, in which the PE first referencing a region of memory will have the page holding that memory assigned to it.

A very common technique in OpenMP program is to initialize data in parallel using the same loop schedule as will be used later in the computations.
NUMA Architecture
Non-Uniform Memory Access

- Data should be on the local memory
  - Local memory: Memory of CPU socket where the core locates.
NUMA Architecture
Non-Uniform Memory Access

- If data are on remote memory, it is not efficient.
NUMA Architecture
Non-Uniform Memory Access

- “First touch data placement” can keep data on local memory.
- “Parallel initialization” of arrays is effective.
use omp_lib
implicit REAL*8 (A-H,O-Z)
real(kind=8), dimension(:,), allocatable :: X, Y
real(kind=8), dimension(:,), allocatable :: Z1, Z2
real(kind=8), dimension(:,), allocatable :: Z3, Z4, Z5
integer, dimension(0:2) :: INDEX

!C
!C +-----+
!C | INIT |
!C +-----+
!C====
write (*,*) 'N, nopt ?'
read  (*,*)  N, nopt
allocate (X(N), Y(N), Z1(N), Z2(N), Z3(N), Z4(N), Z5(N))
if (nopt.eq.0) then
   X = 1.d0
   Y = 1.d0
   Z1= 0.d0
   Z2= 0.d0
   Z3= 0.d0
   Z4= 0.d0
   Z5= 0.d0
else
   !$omp parallel do private (i)
   do i = 1, N
      X (i)= 0.d0
      Y (i)= 0.d0
      Z1(i)= 0.d0
      Z2(i)= 0.d0
      Z3(i)= 0.d0
      Z4(i)= 0.d0
      Z5(i)= 0.d0
   enddo
   !$omp end parallel do
endif
ALPHA= 1.d0

nopt=0
NO First Touch

Initialization is not parallelized
test.f (1/3): Initialization

use omp_lib
implicit REAL*8 (A-H,O-Z)
real(kind=8), dimension(:,), allocatable :: X, Y
real(kind=8), dimension(:,), allocatable :: Z1, Z2
real(kind=8), dimension(:,), allocatable :: Z3, Z4, Z5
integer, dimension(0:2) :: INDEX

!C
| INIT |

!C

write (*,*) 'N, nopt ?'
read  (*,*)  N, nopt
allocate (X(N), Y(N), Z1(N), Z2(N), Z3(N), Z4(N), Z5(N))
if (nopt.eq.0) then
  X = 1.d0
  Y = 1.d0
  Z1= 0.d0
  Z2= 0.d0
  Z3= 0.d0
  Z4= 0.d0
  Z5= 0.d0
else
  !$omp parallel do private (i)
    do i = 1, N
      X (i)= 0.d0
      Y (i)= 0.d0
      Z1(i)= 0.d0
      Z2(i)= 0.d0
      Z3(i)= 0.d0
      Z4(i)= 0.d0
      Z5(i)= 0.d0
    enddo
  !$omp end parallel do
endif
ALPHA= 1.d0

!C

nopt≠0
with First Touch

Initialization is in parallel manner.
Report P1 (1/2)

• Apply multi-threading by OpenMP on parallel FEM code using MPI
  – CG Solver (solver_CG, solver_SR)
  – Matrix Assembling (mat_ass_main, mat_ass_bc)

• Hybrid parallel programming model

• Evaluate the effects of
  – Problem size, parallel programming model, thread #
Report P1 (2/2)

• Deadline: 17:00 October 11th (Sat), 2014.
  – Send files via e-mail at nakajima(at)cc.u-tokyo.ac.jp

• Report
  – Cover Page: Name, ID, and Problem ID (P1)
  – Less than 20 pages including figures and tables (A4).
    • Strategy
    • Structure of the Program
    • Numerical Experiments, Performance Analysis
    • Remarks
  – Source list of the program

• Grade
  – A（優） might be given, if “CG” and “MAT_ASS” are done.
  – B（良） is the highest grade if “MAT_ASS” is NOT done.
```fortran
!$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
```

FORTRAN (solver_CG)
C(solver_CG)

```c
#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];
}
```

```c
#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;
}
```c
for( neib=1; neib<=NEIBPETOT; neib++) {
  istart= EXPORT_INDEX(neib-1);
  inum  = EXPORT_INDEX(neib ) - istart
  #pragma omp parallel for private (k,ii)
  do k= istart+1, istart+inum
    ii   = EXPORT_ITEM(k)
    WS(k)= X(ii)
  enddo
  call MPI_Isend (WS(istart+1), inum, MPI_DOUBLE_PRECISION, &
                           & NEIBPE(neib), 0, MPI_COMM_WORLD, req1(neib), &
                           & ierr)
enddo
```

```c
for( k=istart;k<istart+inum;k++){
  ii = EXPORT_ITEM[k];
  WS[k]= X[ii-1];
}
MPI_Isend(&WS[istart], inum, MPI_DOUBLE,
           NEIBPE[neib-1], 0, MPI_COMM_WORLD, &req1[neib-1]);
```
How to apply multi-threading

• CG Solver
  – Just insert OpenMP directives
  – ILU/IC preconditioning is much more difficult

• MAT_ASS (mat_ass_main, mat_ass_bc)
  – Data Dependency
  – Avoid to accumulate contributions of multiple elements to a single node simultaneously (in parallel)
    • results may be changed
    • deadlock may occur
  – Coloring
    • Elements in a same color do not share a node
    • Parallel operations are possible for elements in each color
    • In this case, we need only 8 colors for 3D problems (4 colors for 2D problems)
    • Coloring part is very expensive: parallelization is difficult
Multi-Threading: Mat_Ass
Parallel operations are possible for elements in same color (they are independent)
allocate (ELMCOLORindex(0:NP))  Number of elements in each color
allocate (ELMCOLORItem (ICELTOT)) Element ID renumbered according to “color”
if (allocated (IWKX)) deallocate (IWKX)
allocate (IWKX(0:NP,3))

IWKX= 0
icou= 0
do icol= 1, NP
  do i= 1, NP
    IWKX(i,1)= 0
  enddo
endo
do icel= 1, ICELTOT
  if (IWKX(icel,2).eq.0) then
    in1= CELNOD(icel,1)
in2= CELNOD(icel,2)
in3= CELNOD(icel,3)
in4= CELNOD(icel,4)
in5= CELNOD(icel,5)
in6= CELNOD(icel,6)
in7= CELNOD(icel,7)
in8= CELNOD(icel,8)
    ip1= IWKX(in1,1)
    ip2= IWKX(in2,1)
    ip3= IWKX(in3,1)
    ip4= IWKX(in4,1)
    ip5= IWKX(in5,1)
    ip6= IWKX(in6,1)
    ip7= IWKX(in7,1)
    ip8= IWKX(in8,1)
isum = ip1 + ip2 + ip3 + ip4 + ip5 + ip6 + ip7 + ip8
if (isum.eq.0) then
   None of the nodes is accessed in the same color
   icou = icou + 1
   IWKX(icol,3) = icou
   (Current) number of elements in each color
   IWKX(icel,2) = icol
   ELMCOLORitem(icou) = icel
   ID of icou-th element = icel

   IWKX(in1,1) = 1
   IWKX(in2,1) = 1
   These nodes on the same elements cannot be
   IWKX(in3,1) = 1
   accessed in the same color
   IWKX(in4,1) = 1
   IWKX(in5,1) = 1
   IWKX(in6,1) = 1
   IWKX(in7,1) = 1
   IWKX(in8,1) = 1
endif
endif
enddo
enddo

100 continue
ELMCOLORtot = icol
Number of Colors
IWKX(0,3) = 0
IWKX(ELMCOLORtot,3) = ICELTOT

   do icol = 0, ELMCOLORtot
      ELMCOLORindex(icol) = IWKX(icol,3)
   enddo
Multi-Threaded Matrix Assembling Procedure

```fortran
  do icol= 1, ELMCOLORtot
    !$omp parallel do private (icel0, icel)
      &$omp&      private (in1, in2, in3, in4, in5, in6, in7, in8)
      &$omp&      private (nodLOCAL, ie, je, ip, jp, kk, iiS, iiE, k)
      &$omp&      private (DETJ, PNX, PNY, PNZ, QVC, QVO, COEFij, coef, SHi)
      &$omp&      private (PNXi, PNYi, PNZi, PNXj, PNYj, PNZj, ipn, jpn, kpn)
      &$omp&      private (X1, X2, X3, X4, X5, X6, X7, X8)
      &$omp&      private (Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8)
      &$omp&      private (Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8, CONDO)
    do icel0= ELMCOLORindex(icol-1)+1, ELMCOLORindex(icol)
      icel= ELMCOLORitem(icel0)
      in1= ICELNOD(icel,1)
      in2= ICELNOD(icel,2)
      in3= ICELNOD(icel,3)
      in4= ICELNOD(icel,4)
      in5= ICELNOD(icel,5)
      in6= ICELNOD(icel,6)
      in7= ICELNOD(icel,7)
      in8= ICELNOD(icel,8)
      ...
```
## Results (1/2)

512 × 384 × 256 = 50,331,648 nodes
12 nodes, 192 cores
64³=262,144 nodes/core

<table>
<thead>
<tr>
<th>ndx, ndy, ndz (#MPI proc.)</th>
<th>Iter’s</th>
<th>sec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat MPI</td>
<td>8 6 4 (192)</td>
<td>1240</td>
</tr>
<tr>
<td>HB 1 × 16</td>
<td>8 6 4 (192)</td>
<td>1240</td>
</tr>
<tr>
<td>HB 2 × 8</td>
<td>4 6 4 (96)</td>
<td>1240</td>
</tr>
<tr>
<td>HB 4 × 4</td>
<td>4 3 4 (48)</td>
<td>1240</td>
</tr>
<tr>
<td>HB 8 × 2</td>
<td>4 3 2 (24)</td>
<td>1240</td>
</tr>
<tr>
<td>HB 16 × 1</td>
<td>2 3 2 (12)</td>
<td>1240</td>
</tr>
</tbody>
</table>
## Results (2/2)

512 × 384 × 256 = 50,331,648 nodes  
12 nodes, 192 cores  
64³=262,144 nodes/core

<table>
<thead>
<tr>
<th>OMP_NUM_THREADS</th>
<th>sec.</th>
<th>Speed-Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1056.2</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>592.5</td>
<td>1.78</td>
</tr>
<tr>
<td>4</td>
<td>289.8</td>
<td>3.64</td>
</tr>
<tr>
<td>8</td>
<td>148.1</td>
<td>7.13</td>
</tr>
<tr>
<td>12</td>
<td>103.6</td>
<td>10.19</td>
</tr>
<tr>
<td>16</td>
<td>81.9</td>
<td>12.90</td>
</tr>
<tr>
<td>Flat MPI, 1 proc./node</td>
<td>1082.4</td>
<td>-</td>
</tr>
</tbody>
</table>
Flat MPI vs. Hybrid

- Depends on applications, problem size, HW etc.
- Flat MPI is generally better for sparse linear solvers, if number of computing nodes is not so large.
  - Memory contention
- Hybrid becomes better, if number of computing nodes is larger.
  - Fewer number of MPI processes.
- Flat MPI is not realistic for Intel Xeon Phi/MIC with 240 threads/node
  - MPI process requires certain amount of memory.
omp parallel (do)

• “omp parallel-omp end parallel” generates/eliminates threads at every call: fork-join
• This could be overhead for multiple loops
• omp parallel + omp do/omp for

```c
!$omp parallel ...

!$omp do
do i = 1, N
...

!$omp do
do i = 1, N
...

!$omp end parallel 必須
```

```
#pragma omp parallel ...

#pragma omp for {
...

#pragma omp for {
```
Tuning

- List of Messages by Compiler (Compile List)
- 精密PA可視化機能 (Excel)
  (Precision PA Visibility Function)
List of Messages by Compiler (Compile List)

F90 = mpifrtpx
F90OPTFLAGS = -Kfast,openmp -Qt
F90FLAGS = $(F90OPTFLAGS)

- **-Qt**
  - List of Messages by Compiler (Compile List)
  - *.lst
  - Fortran Only

- In C, "-Qt" is not available
  - Please use "-Nsfc"
  - Displayed on screen
Current version of C/C++ compiler can produce list of messages

<table>
<thead>
<tr>
<th>Fortran/C/C++</th>
<th>方言の最適化情報（デフォルト）</th>
</tr>
</thead>
<tbody>
<tr>
<td>-N1st=p</td>
<td>標準の最適化情報</td>
</tr>
<tr>
<td>-N1st=t</td>
<td>詳細な最適化情報</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fortran ONLY</th>
<th>名前の属性情報</th>
</tr>
</thead>
<tbody>
<tr>
<td>-N1st=a</td>
<td>名前の属性情報</td>
</tr>
<tr>
<td>-N1st=d</td>
<td>派生型の構成情報</td>
</tr>
<tr>
<td>-N1st=i</td>
<td>インクルードされたファイルのプログラムリストおよびインクルードファイル名一覧</td>
</tr>
<tr>
<td>-N1st=m</td>
<td>自動並列化の状況をOpenMP指示文によって表現した原始プログラム出力</td>
</tr>
<tr>
<td>-N1st=x</td>
<td>名前および文番号の相互参照情報</td>
</tr>
</tbody>
</table>
Info in *.lst

ループ単位の最適化情報
- ループ最適化情報
  (ループ融合、ループ交換等)
- SIMD化情報
- 並列化情報
- ソフトウェアパイプラインング
- プリフェッチ情報

ライン単位の最適化情報
- SIMD化情報
- 並列化情報
- ループアンローリング展開数
- インライン展開情報

```
(line-no.)(nest)(optimize)
1  subroutine sub(a, b, n)
2    real*8 a(n), b(n)
<<<< Loop-information Start >>>>
<<<< [PARALLELIZATION]
<<<< Standard iteration count: 800
<<<< [OPTIMIZATION]
<<<< SIMD
<<<< SOFTWARE PIPELINING
<<<< Loop-information End >>>>
3  i p 8v  do i=1,n
4  1 p 8v     a(i)=b(i)+b(i+1)+b(i+2)
5  1 p 8v     enddo
6  1 p 8v     end

Procedure information
  Lines   : 6
  Statements : 6
  Stack(byte): 64
  Prefetch num: 0
```
SIMD Information

DOループのSIMD化情報
v: SIMD化された
m: SIMD化された部分とSIMD化されなかった部分を含む
s: SIMD化されなかった
空白: SIMD化対象でない

実行文のSIMD化情報
v: SIMD化可能
m: SIMD化可能な部分とSIMD化不可能な部分を含む
s: SIMD化不可能

(line-no.) (nest) (optimize)
2  1
3  1
doj = 1, n
<<< Loop-information Start >>>
<<< [OPTIMIZATION]
<<< SIMD
<<< Loop-information End >>>
4  2  do i = 1, n
5  2  a(i,j) = b(i,j)*2.0
6  2  enddo
7  1  enddo

:
Automatic Parallelization

DOループの並列化情報
pp : 並列化された
m : 並列化された部分と並列化されなかった部分を含む
s : 並列化されなかった
空白 : 並列化対象でない

ループの繰返し数が
1778以上の時は並列実行
1778未満の時は逐次実行

実行文の並列化情報
p : 並列化可能
m : 並列化可能な部分と並列化不可能な部分を含む
s : 並列化不可能
Example of the List

```
101  1  !C
102  1  !C +------------------+
103  1  !C \{z\} = [Minv]\{r\} |
104  1  !C +------------------+
105  1  !C==
106  1  
107  1  !$omp parallel do private(ip,i)
108  2  p  do ip = 1, PEsmpTOT
109  3  p  8v  do i = SMPindexG(ip-1)+1, SMPindexG(ip)
110  3  p  8v  W(i,Z)= W(i,R)
111  3  p  8v  enddo
112  2  p  enddo
113  1  !$omp end parallel do
114  1  
115  1  Stime= omp_get_wtime()
116  1  call fapp_start ("precond", 1, 1)
117  2  do ic= 1, NCOLORtot
118  2  !$omp parallel do private(ip,ip1,i,WVAL,k)
119  3  p  do ip = 1, PEsmpTOT
120  3  p  ip1= (ic-1)*PEsmpTOT + ip
121  4  p  do i = SMPindex(ip1-1)+1, SMPindex(ip1)
122  4  p  WVAL= W(i,Z)
123  5  p  4v  do k= indexL(i-1)+1, indexL(i)
124  5  p  4v  WVAL= WVAL - AL(k) * W(itemL(k),Z)
125  5  p  4v  enddo
126  4  p  W(i,Z)= WVAL * W(i,DD)
127  4  p  enddo
128  3  p  enddo
129  2  !$omp end parallel do
130  2  enddo
```
3.5 Precision PA Visibility Function (Excel)

(Precision PA Visibility Function)

(1/3): Inserting Call’s, Compile & Run

call start_collection ("SpMV")
!$omp parallel do private(ip,i,VAL,k)
do ip= 1, PEsmpTOT
do i= SMPindex((ip-1)*NCOLORtot)+1, SMPindex(ip*NCOLORtot)
VAL= D(i)*W(i,P)
do k= 1, 3
VAL= VAL + AL(k,i)*W(itemL(k,i),P)
enddo
do k= 1, 3
VAL= VAL + AU(k,i)*W(itemU(k,i),P)
enddo
W(i,Q)= VAL
enddo
enddo
!$omp end parallel do
call stop_collection ("SpMV")
3.5 精密PA可視化機能 (Excel) (Precision PA Visibility Function) (2/3): Collecting Performance Data: 7X Exec’s Directories: pa1~pa7, -Hpa=1~7

#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:05:00"
#PJM -L "rscgrp=lecture"
#PJM -g "gt71"
#PJM -j
#PJM -o "3.lst"
#PJM --mpi "proc=1"

export OMP_NUM_THREADS=16
fapp -C -d pa1 -Ihwm -Hpa=1 ./sol-r3k
3.5 精密PA可視化機能 (Excel)  
(Precision PA Visibility Function)  

```bash
fapppx -A -d pa1 -o output_prof_1.csv -tcsv -Hpa
fapppx -A -d pa2 -o output_prof_2.csv -tcsv -Hpa
fapppx -A -d pa3 -o output_prof_3.csv -tcsv -Hpa
fapppx -A -d pa4 -o output_prof_4.csv -tcsv -Hpa
fapppx -A -d pa5 -o output_prof_5.csv -tcsv -Hpa
fapppx -A -d pa6 -o output_prof_6.csv -tcsv -Hpa
fapppx -A -d pa7 -o output_prof_7.csv -tcsv -Hpa
```
Fujitsu FX10: CASE-1, CM-RCM(2)
L1-dem.-miss: 25.6%, Mem. throughput: 41.8GB/sec.
Forward/Backward Substitution

src0: CRS, Coalesced
Fujitsu FX10: CASE-2, CM-RCM(2)
25.6%, 41.8GB/sec.

reorder0: CRS, Sequential
Fujitsu FX10: CASE-3, CM-RCM(2)
5.4%, 64.0GB/sec.

ELL, Sequential