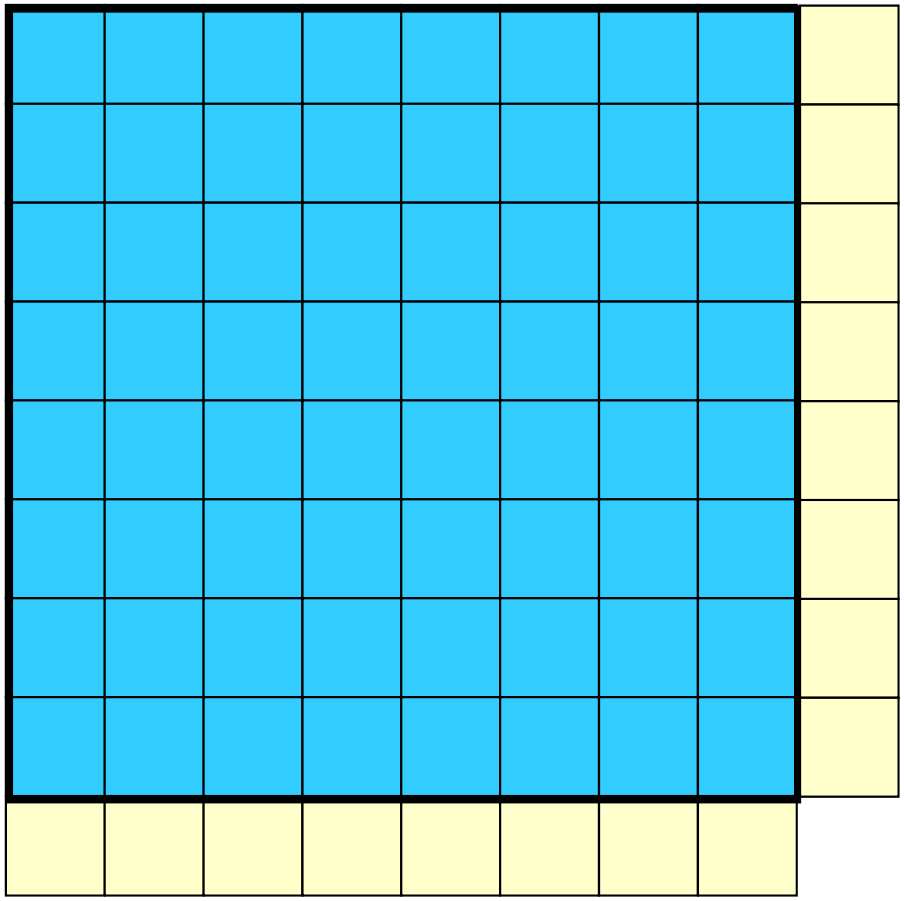
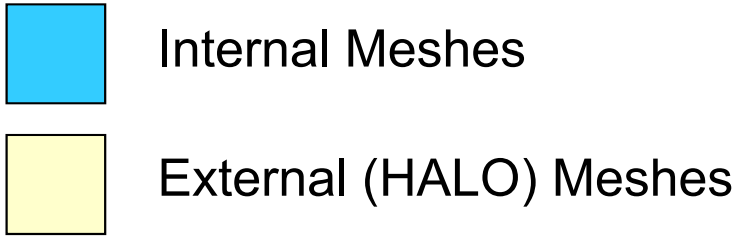
Introduction to Parallel Programming for Multicore/Manycore Clusters

Part II-5: Communication-Computation Overlapping and Some Other Issues

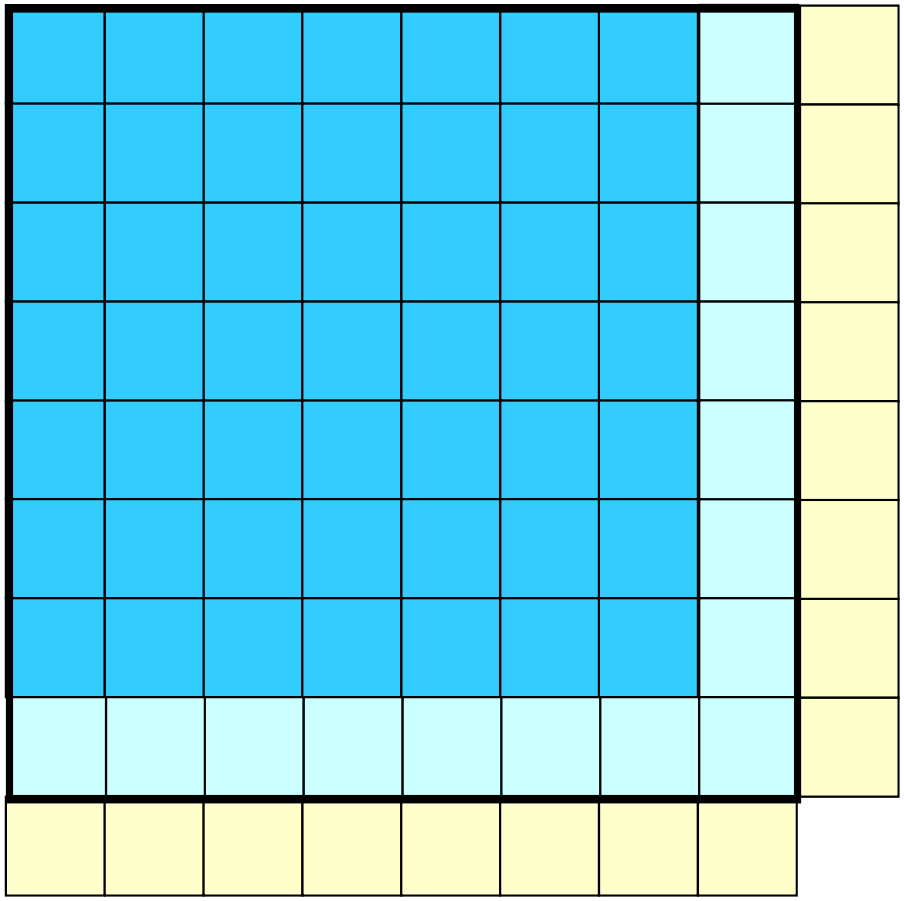
Kengo Nakajima
Information Technology Center
The University of Tokyo

Communication-Computation Overlapping

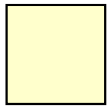




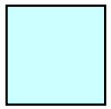
Communication-Computation Overlapping



(Pure) Internal Meshes



External (HALO) Meshes



Internal Meshes on Boundary's

Mat-Vec Multiplication Operations

- Overlapping of computations of internal meshes, and importing external meshes.
- Then computation of international meshes on boundary's
- Difficult for IC/ILU on Hybrid
- Renumbering needed for higher performance

Comm.-Comp. Overlapping

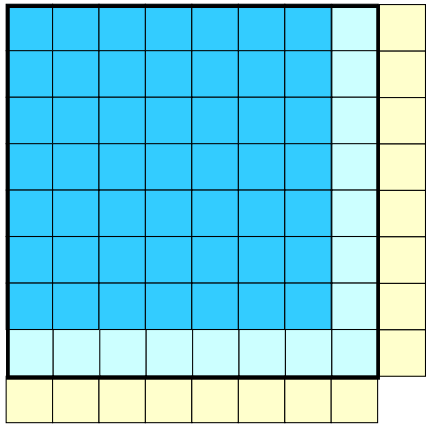
<$O-fvm/src1> Without Reordering

```
MPI_Isend;
MPI_Irecv;

for(i=0;i<Nall;i++){
  if (BOUNDARY[i]==0){
    (calculations)
  }
}

MPI_Waitall;

for(i=0;i<Nall;i++){
  if (BOUNDARY[i]==1){
    (calculations)
  }
}
```

<$O-fvm/src2> With Reordering

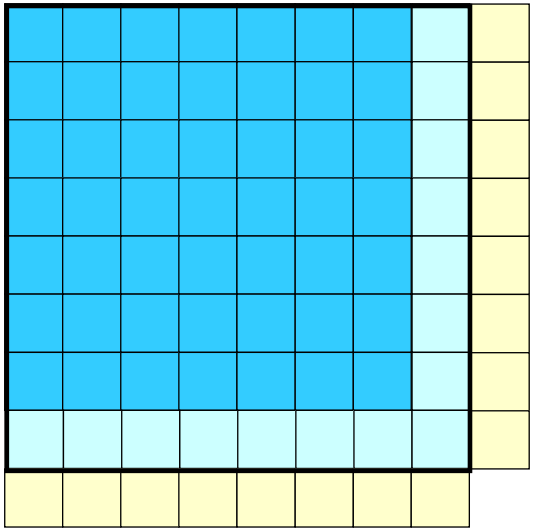
```
MPI_Isend;
MPI_Irecv;

for(i=0;i<Ninn;i++){
  (calculations)
}

MPI_Waitall;

for(i=Ninn;i<Nall;i++){
  (calculationas)
}
```

Communications and computation for pure internal meshes are overlapped. Therefore, problem size should be large enough for *hiding* communication overhead



poi_gen (<$src1>)

```
#pragma omp parallel for private(i)

for(i=0; i<N; i++) {
    for(k=indexLU[i]; k<indexLU[i+1]; k++) {
        jj = itemLU[k];
        if(jj > N) BOUNDARY[i] = 1;
    }
}
```



If the mesh “i” is connected to “external” meshes , the mesh is internal mesh on the boundary .

Finally “BOUDARY(i)” is set to “1”.

[A]{p}={q} (<$src1>)

```
      for(neib=0;neib<NEIBPETOT;neib++){
(...)
          MPI_Isend (...);
      }

      for(neib=0;neib<NEIBPETOT;neib++){
(...)
          MPI_Irecv (...);
      }
```

```
#pragma omp parallel for private(i,k,VAL)
for(i=0; i<N; i++) {
    if(BOUNDARY[i] == 0) {
       VAL = D[i] * W[P][i];
       for(k=indexLU[i]; k<indexLU[i+1]; k++) {
           VAL += AMAT[k] * W[P][itemLU[k]-1];
       }
       W[Q][i] = VAL;
    }
}
```

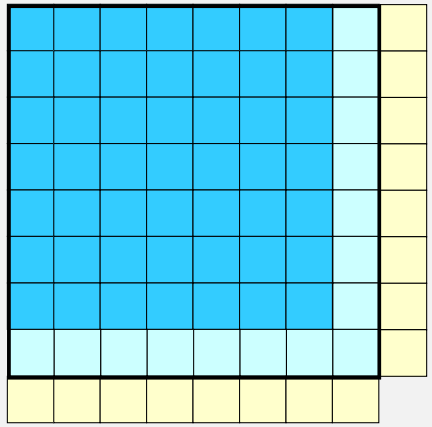
Pure Internal Meshes

Overlapping

```
MPI_WAITALL (...);
```

```
#pragma omp parallel for private(i,k,VAL)
for(i=0; i<N; i++) {
    if(BOUNDARY[i] == 0) {
       VAL = D[i] * W[P][i];
       for(k=indexLU[i]; k<indexLU[i+1]; k++) {
           VAL += AMAT[k] * W[P][itemLU[k]-1];
       }
       W[Q][i] = VAL;
    }
}
```

Boundary Meshes



Communications and computation for pure internal meshes are overlapped. Therefore, problem size should be large enough for *hiding* communication overhead

poi_gen (<$src2>)

```
#pragma omp parallel for private (icel,icN1,icN2,icN3,icN4,icN5,icN6)
for(icel=0;icel<ICELTOT;icel++){
    icN1= NEIBcell[icel][0] ...
    if (icN1 > ICELTOT) BNODE[icel]= 1;
    if (icN2 > ICELTOT) BNODE[icel]= 1;
    if (icN3 > ICELTOT) BNODE[icel]= 1; ...
}
```

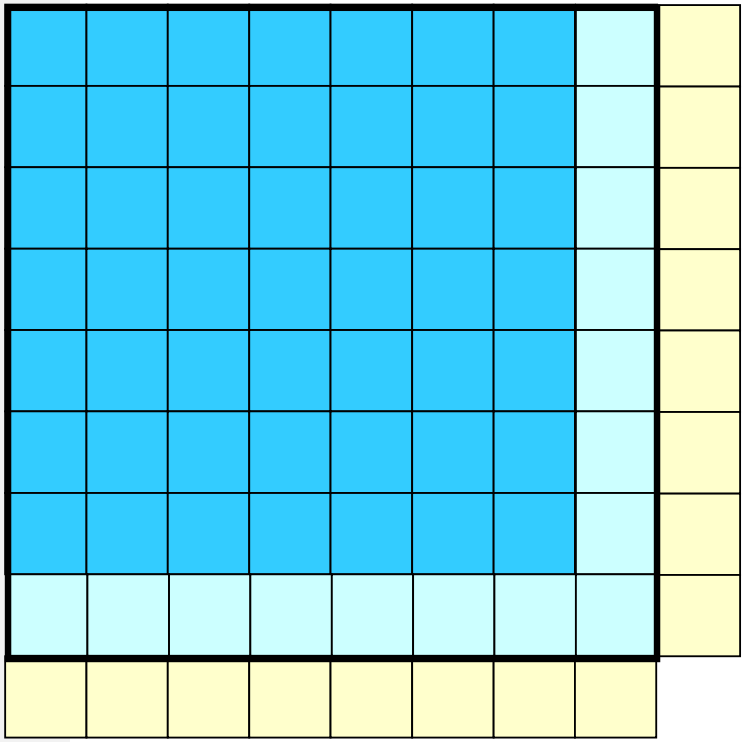
BNODE[icel]=1
if “icel” is on bounday’s

```
      icou= 0
      for(ice=0;icel<ICELTOT;icel++)
        if (BNODE(icel).==0) {
          OLDtoNEW[icel]= icou+1;
          NEWtoOLD[icou]= icel+1;
          icou++;
        }
      }
      ICELTOTinn= icou;
      icou= 0
      for(ice=0;icel<ICELTOT;icel++)
        if (BNODE(icel).==0) {
          OLDtoNEW[icel]= icou+1;
          NEWtoOLD[icou]= icel+1;
          icou++;
        }
      }

for(neib=0; neib<NEIBPETOT; neib++) {
    for(k=EXPORT_INDEX[neib]; k<EXPORT_INDEX[neib+1]; k++) {
        BNODE[k] = EXPORT_ITEM[k];
    }
}

for(neib=0; neib<NEIBPETOT; neib++) {
    for(k=EXPORT_INDEX[neib]; k<EXPORT_INDEX[neib+1]; k++) {
        EXPORT_ITEM[k] = OLDtoNEW[BNODE[k]-1];
    }
}
```

Renumbering
Pure Internal Meshes
ICELTOTinn

Renumbering
Boundary Meshes

Renumbering
EXPORT_ITEM

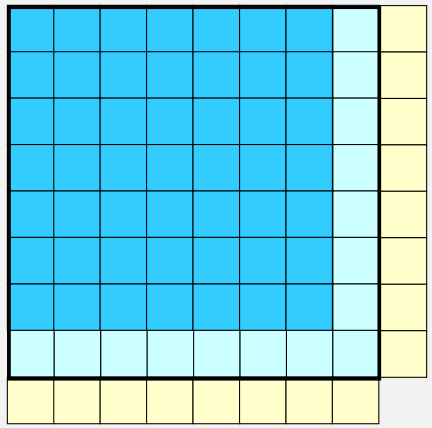


Pure Internal Meshes

Internal Meshes on Boundary’s

[A]{p}={q} (<$src2>)

```
      for(neib=0;neib<NEIBPETOT;neib++){
(...)
          MPI_Isend (...);
      }

      for(neib=0;neib<NEIBPETOT;neib++){
(...)
          MPI_Irecv (...);
      }
```



```
#pragma omp parallel for private(i,k,VAL)
for(i=0; i<Ninn; i++) {
    VAL = D[i] * W[P][i];
    for(k=indexLU[i]; k<indexLU[i+1]; k++) {
        VAL += AMAT[k] * W[P][itemLU[k]-1];
    }
    W[Q][i] = VAL;
}
```

Pure Internal Meshes

Overlapping

```
MPI_WAITALL (...);
```

```
#pragma omp parallel for private(i,k,VAL)
for(i=Ninn; i<N; i++) {
    VAL = D[i] * W[P][i];
    for(k=indexLU[i]; k<indexLU[i+1]; k++) {
        VAL += AMAT[k] * W[P][itemLU[k]-1];
    }
    W[Q][i] = VAL;
}
```

Boundary Meshes

Communications and computation for pure internal meshes are overlapped. Therefore, problem size should be large enough for *hiding* communication overhead

Compile & Run (Hybrid Only)

<$src1>

```
>$ cd
>$ cd hybrid/fvm/src1
>$ make clean
>$ make
>$ ls ../run/sol1-mpih
   sol1-mpi
>$ cd ../run
(modify go1.sh, INPUT.DAT)
>$ pjsub go1.sh (or mpiexec ...)
```

<$src2>

```
>$ cd
>$ cd hybrid/fvm/src2
>$ make clean
>$ make
>$ ls ../run/sol2-mpih
   sol2-mpi
>$ cd ../run
(modify go2.sh, INPUT.DAT)
>$ pjsub go2.sh (or mpiexec ...)
```

Results on Oakleaf-FX 64 nodes, 64 processes, HB 16x1 Time for CG Solvers (sec.) (Fortran)

DOF (NAx)	<$O-fvm>/src Original	<$O-fvm>/src0 Continuous Access	<$O-fvm>/src1 Comm-Comp Overlapping (if-then-branch)	<$O-fvm>/src2 Comm-Comp Overlapping (reordered)
508^3	**33.0**	**33.0**	**38.7**	**34.0**
512^3	78.1	78.0	87.3	36.3
516^3	35.7	35.5	41.7	46.8
636^3	82.4	82.0	93.9	84.4
640^3	109.6	109.4	127.4	88.8
644^3	85.0	85.1	97.3	87.5
744^3	**160.0**	**159.8**	**182.5**	**154.9**

Bank conflict occurs at 512^3 & 640^3 In <$src2> it is avoided due to renumbering

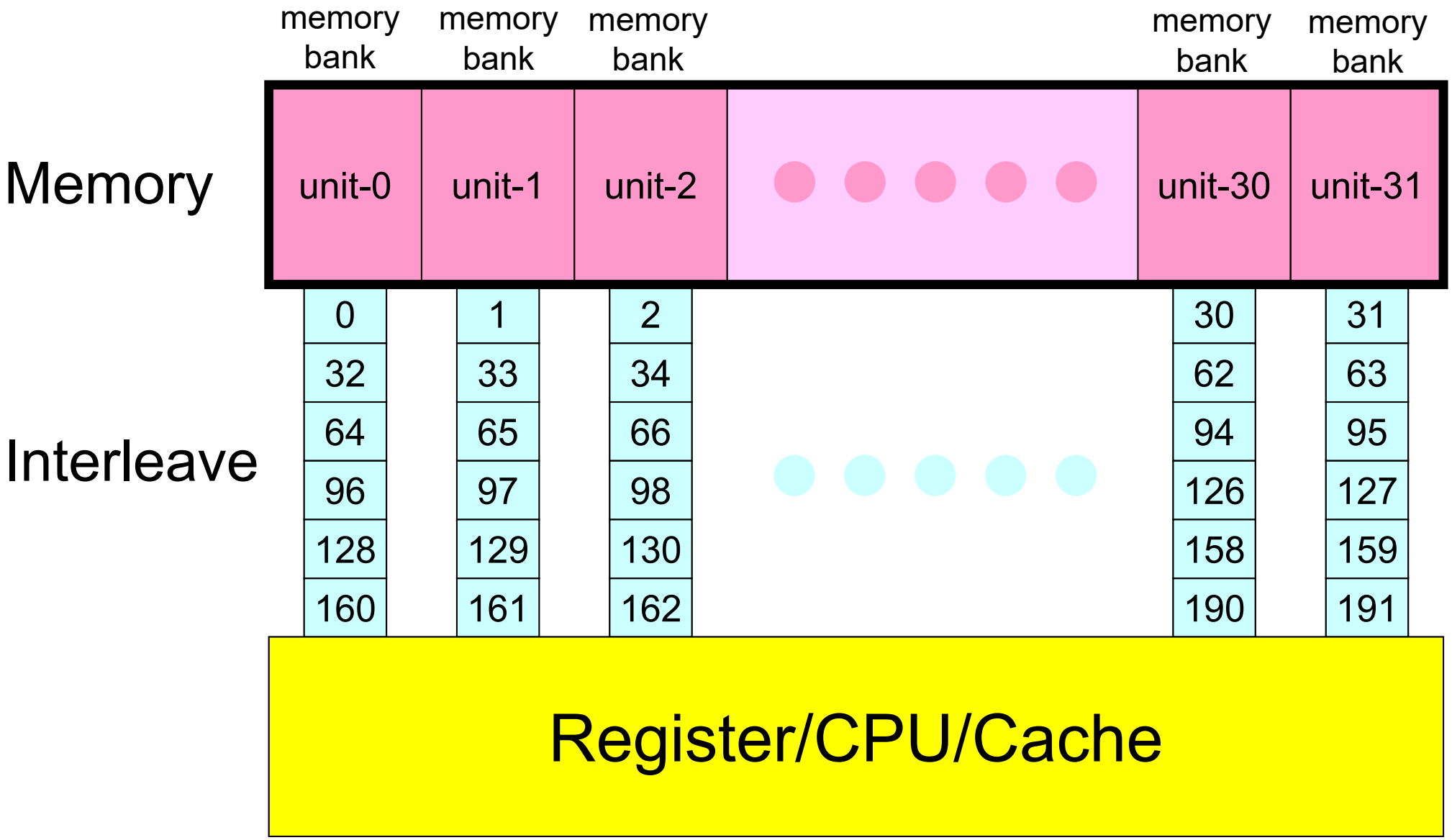
DOF (NAx)	<$O-fvm>/src Original	<$O-fvm>/src0 Continuous Access	<$O-fvm>/src1 Comm-Comp Overlapping (if-then-branch)	<$O-fvm>/src2 Comm-Comp Overlapping (reordered)
508^3	33.0	33.0	38.7	34.0
512^3	78.1	78.0	87.3	36.3
516^3	35.7	35.5	41.7	46.8
636^3	82.4	82.0	93.9	84.4
640^3	109.6	109.4	127.4	88.8
644^3	85.0	85.1	97.3	87.5
744^3	160.0	159.8	182.5	154.9

Memory Interleaving/Bank Conflict

- Memory Interleaving
  - Method for fast data transfer to/from memory.
  - Parallel I/O for multiple memory banks.
- Memory Bank
  - Unit for memory management, small pieces of memory
    - Usually, there are 2^n independent modules.
  - Single bank can execute a single reading or writing at one time. Therefore, performance gets worse if data components on same bank are accessed simultaneously.
  - For example, “bank conflict” occurs if off-set of data access is 32 (in next page).
    - Remedy: Change of array size, loop exchange, reordering etc.

Bank Conflict

If off-set of data access is 32, only a single bank is utilized



Avoiding Bank Conflict

×

```
REAL*8 A(32,10000)

k= N
do i= 1, 10000
   A(k,i)= 0.d0
enddo
```

○

```
REAL*8 A(33,10000)

k= N
do i= 1, 10000
   A(k,i)= 0.d0
enddo
```

- Arrays with size of 2^n should be avoided.

If N=2^m, e.g. 8x8

- Bank conflict always occurs when non-zero off-diagonals are accessed, 16 threads
- Remedy for Bank Conflict
  - Padding by compiler, Reordering

73	74	75	76	77	78	79	80	
57	58	59	60	61	62	63	64	72
49	50	51	52	53	54	55	56	71
41	42	43	44	45	46	47	48	70
33	34	35	36	37	38	39	40	69
25	26	27	28	29	30	31	32	68
17	18	19	20	21	22	23	24	67
9	10	11	12	13	14	15	16	66
1	2	3	4	5	6	7	8	65

73	74	75	76	77	78	79	80	
57	58	59	60	61	62	63	64	72
43	44	45	46	47	48	49	56	71
36	37	38	39	40	41	42	55	70
29	30	31	32	33	34	35	54	69
22	23	24	25	26	27	28	53	68
15	16	17	18	19	20	21	52	67
8	9	10	11	12	13	14	51	66
1	2	3	4	5	6	7	50	65

If you want to do these on Oakleaf-FX

by 12 nodes, 12 MPI processes, HB 16x1

128^3 for each node (MPI process)

mg.sh

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L “rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi "proc=12"

mpiexec ./pmesh
```

mesh.inp

```
384  256  256
  3    2    2
```

goh.sh

```
#!/bin/sh
#PJM -L "node=12"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L “rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi "proc=12"

export OMP_NUM_THREADS=16
mpiexec ./sol-mpih
```

Cache Thrashing

- FX10: L1D cache with 32KB for each core, 2-way
  - n-way set associative（n群連想記憶式）
  - Cache is divided into “n” banks
  - Each bank is divided into “cache lines”
    - Number of Cache Lines, Size of Cache Line (128 bytes for FX10) ⇒ 2^m
- This “2-way” cache is very harmful
  - If “$N=2^m$“, memory addresses of `W(i,P)`, `W(i,Q)`, `W(i,R)` map to the same cache address in CG computation.
  - Cache Thrashing: Lower Performance
    - `R=1, P=2, Q=3`
    - `X(i)` is not affected

```
!$omp parallel do private(i)
      do i= 1, N
        X(i)   = X(i)   + ALPHA * W(i,P)
        W(i,R)= W(i,R) - ALPHA * W(i,Q)
      enddo
```

Remedy

- If the loop is split into 2 loops, up to 2 cache lines of W are referred. Therefore, cache thrashing does not occur (Remedy-1).

```
!$omp parallel do private(i)
      do i= 1, N
         X(i)  = X (i)   + ALPHA * W(i,P)
      enddo

!$omp parallel do private(i)
      do i= 1, N
        W(i,R)= W(i,R) - ALPHA * W(i,Q)
      enddo
```

- If “N=2^m“, certain numbers (e.g. 64, 128 …) can be added to N. Thus, size of the array is not equal to 2^m, and cache thrashing is avoided (Remedy-2).
  - No such operation is needed for X

```
N2=128
allocate (W(N+N2,4))
```

Remedy-2: Try by yourself <$src2>

solver_PCG: C

```
N3= N+N2;                                                    N2=128
W = (double **)malloc(sizeof(double *)*4);
…
for(i=0; i<4; i++) {
    W[i] = (double *)malloc(sizeof(double)*N3);
…
```

solver_PCG: Fortran

```
allocate (W(N+N2,4))                                         N2=128
```

Results Fortran

	NX=NY=NZ=128 2,097,152 meshes Load/Store= 8.28×10^{10}	NX=NY=NZ=129 2,146,689 meshes Load/Store= 8.53×10^{10}
sol10 (original)	19.50 sec. 24.11 GB/sec 13.59 %	9.15 sec. 52.98 GB/sec 3.97 %
sol20 (CM reordering)	10.15 sec. 45.60 GB/sec 5.65 %	9.44 sec. 50.64 GB/sec 4.11 %
sol2x (CM+ Remedy-2)	9.69 sec. 47.77 GB/sec 4.20 %	9.54 sec. 50.12 GB/sec 4.11 %

C ($N=128^3$)

- sol10: 20.03 sec.
- sol20: 13.72
- sol2x: 10.05

- Comp. Time
- Memory Throughput
- L1D Miss Ratio (to Load/Store)

Cache Thrashing, Bank Conflict

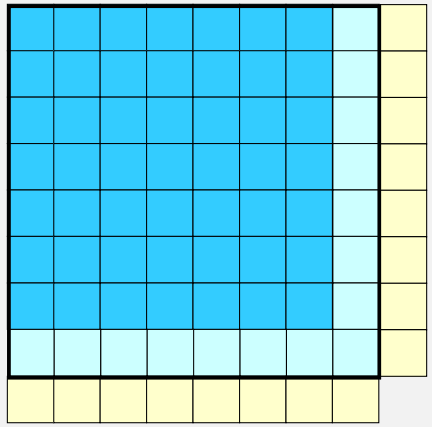
- Problem is that the cache is 2-way on FX10
  - This is the reason for both of bank conflict and cache thrashing
- Most of modern architectures based on 4-way or 8-way
  - e.g. Intel CPU
- FX-100 (successor of FX10) has 4-way cache

Back to "Communication-Computation Overlapping"

- Problem size should be large enough for hiding communications by overlapping.
- Number of compute nodes should be large enough (too small in this case)
- Generally speaking, effect of communication/computation overlapping is rather smaller for Krylov iterative solvers
  - Explicit time-marching method by FDM

[A]{p}={q} (<$src2>)

```
      for(neib=0;neib<NEIBPETOT;neib++){
(...)
          MPI_Isend (...);
      }

      for(neib=0;neib<NEIBPETOT;neib++){
(...)
          MPI_Irecv (...);
      }
```



```
#pragma omp parallel for private(i,k,VAL)
for(i=0; i<Ninn; i++) {
    VAL = D[i] * W[P][i];
    for(k=indexLU[i]; k<indexLU[i+1]; k++) {
        VAL += AMAT[k] * W[P][itemLU[k]-1];
    }
    W[Q][i] = VAL;
}
```

Pure Internal Meshes

Overlapping

```
MPI_WAITALL (...);
```

```
#pragma omp parallel for private(i,k,VAL)
for(i=Ninn; i<N; i++) {
    VAL = D[i] * W[P][i];
    for(k=indexLU[i]; k<indexLU[i+1]; k++) {
        VAL += AMAT[k] * W[P][itemLU[k]-1];
    }
    W[Q][i] = VAL;
}
```

Boundary Meshes

Communications and computation for pure internal meshes are overlapped. Therefore, problem size should be large enough for *hiding* communication overhead

Bank conflict occurs at 512^3 & 640^3 In <$src2> it is avoided due to renumbering

DOF (NAx)	<$O-fvm>/src Original	<$O-fvm>/src0 Continuous Access	<$O-fvm>/src1 Comm-Comp Overlapping (if-then-branch)	<$O-fvm>/src2 Comm-Comp Overlapping (reordered)
508^3	33.0	33.0	38.7	34.0
512^3	78.1	78.0	87.3	36.3
516^3	35.7	35.5	41.7	46.8
636^3	82.4	82.0	93.9	84.4
640^3	109.6	109.4	127.4	88.8
644^3	85.0	85.1	97.3	87.5
744^3	160.0	159.8	182.5	154.9