Introduction to Parallel Programming for Multicore/Manycore Clusters

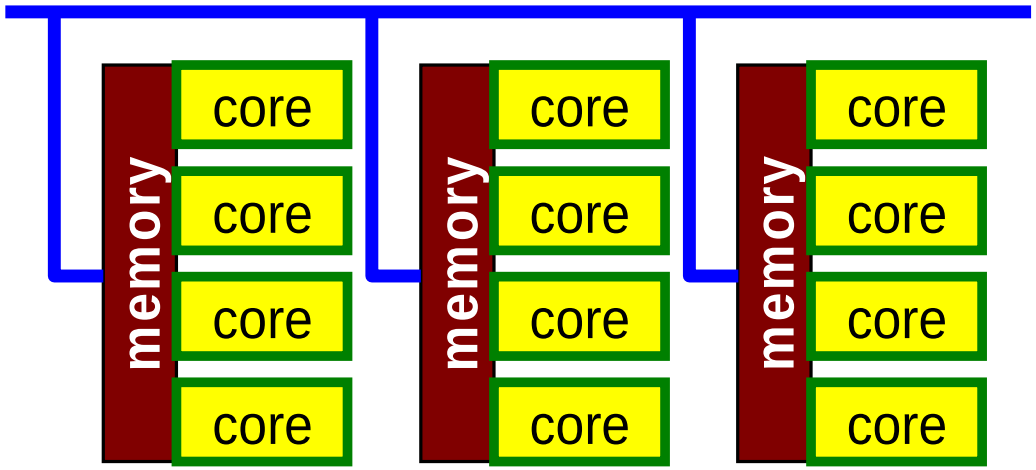
Part II: Introduction to OpenMP

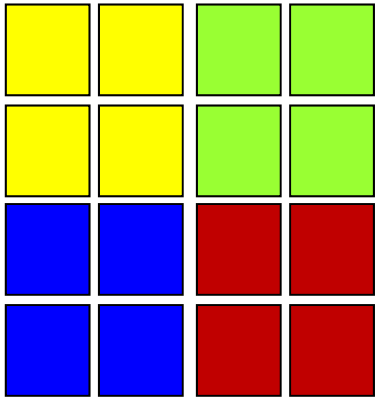
Kengo Nakajima
Information Technology Center
The University of Tokyo

- OpenMP
- Login to FX10
- Parallel Version of the Code by OpenMP
- STREAM
- Using the Profiler of FX10
- Data Dependency

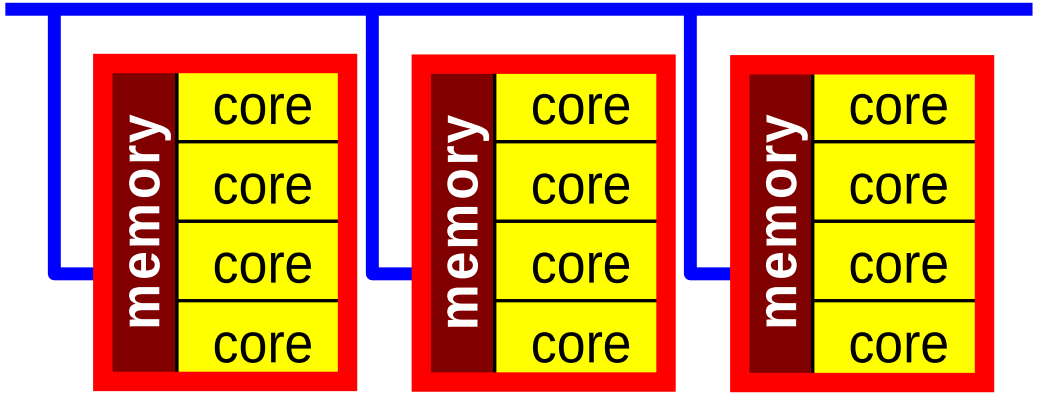
Flat MPI vs. Hybrid

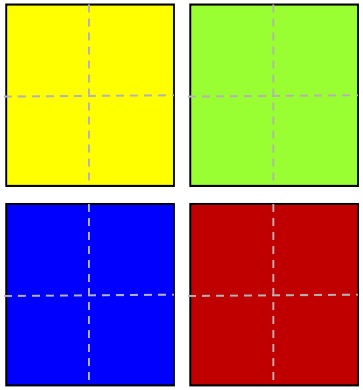
Flat-MPI：Each Core -> Independent

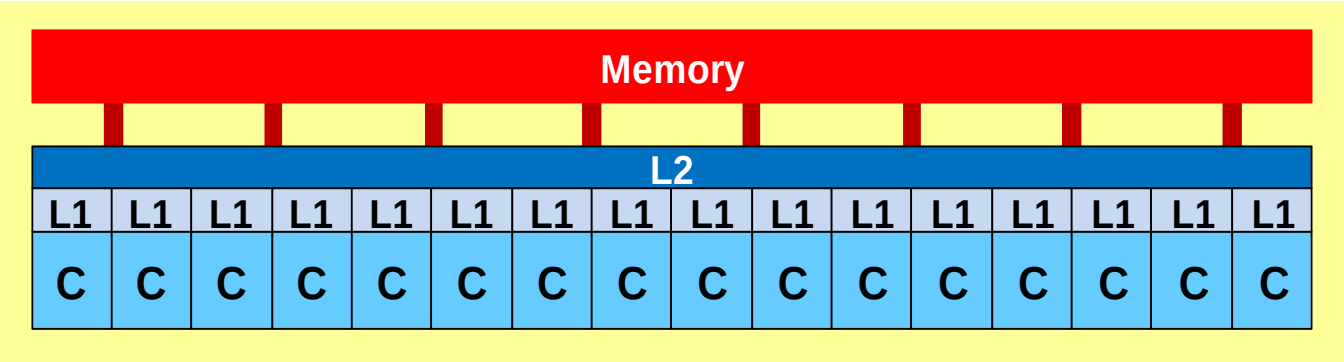




Hybrid：Hierarchal Structure







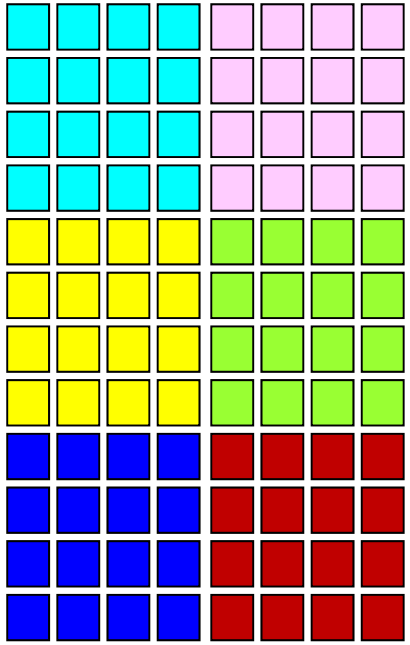
HB M x N

Number of OpenMP threads per a single MPI process

Number of MPI process per a single node

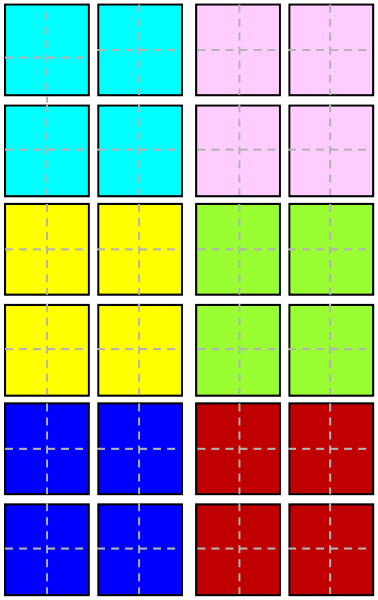
Size of data for each MPI process varies according to HB MxN

example: 6 nodes, 96 cores



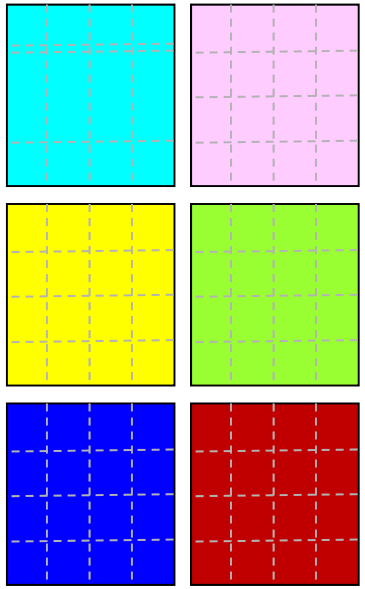
Flat MPI

```
128  192  64
  8   12   1
pcube
```



HB 4x4

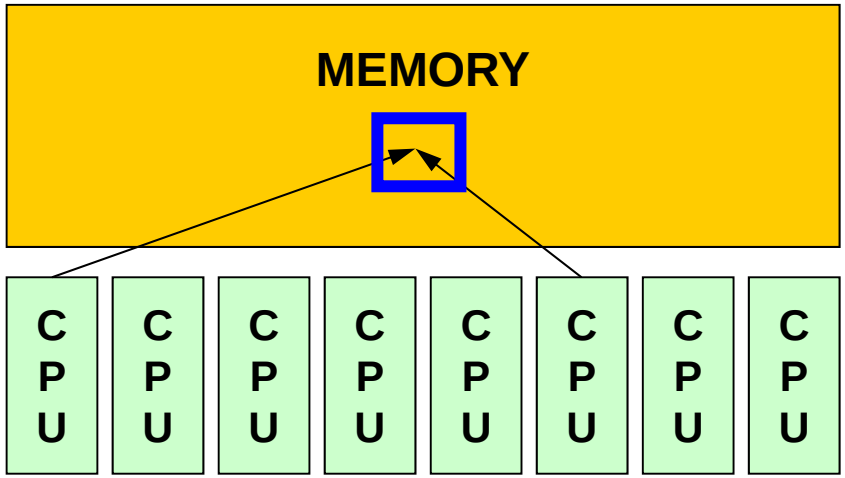
```
128  192  64
  4    6   1
pcube
```



HB 16x1

```
128  192  64
  2    3   1
pcube
```

SMP



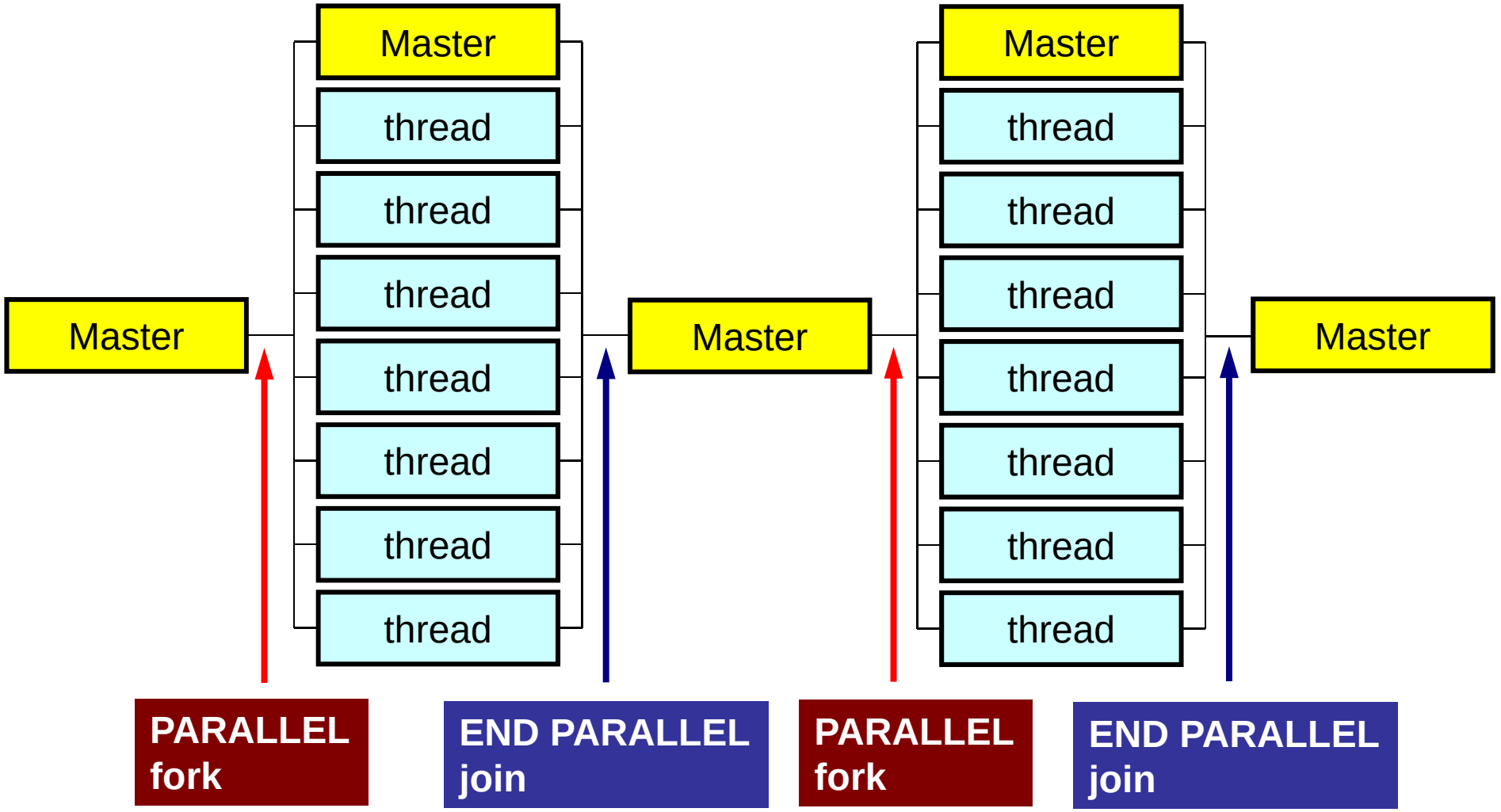
- SMP
  - Symmetric Multi Processors
  - Multiple CPU’s (cores) share a single memory space

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: GPU, Intel-MIC: similar to OpenACC
- 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）
  - Chapman, B. et al.「Using OpenMP」（MIT Press）
- Japanese Version of OpenMP 3.0 Spec. (Fujitsu etc.)
  - http://www.openmp.org/mp-documents/OpenMP30spec-ja.pdf

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

```
#pragma omp parallel for private (i)
 for (i=0; i<N; i++) {
      X[i] = 0.0;
      W[0][i] = 0.0;
      W[1][i] = 0.0;
      W[2][i] = 0.0;
 }
```

Dot Products

```
RHO = 0.0;
#pragma omp parallel for private (i)
 reduction (+:RHO)
 for(i=0; i<N; i++) {
     RHO += W[R][i] * W[Z][i];
 }
```

DAXPY

```
#pragma omp parallel for private (i)
 for (i=0; i<N; i++) {
    Y[i] = Y[i} + alpha*X[i];
 }
```

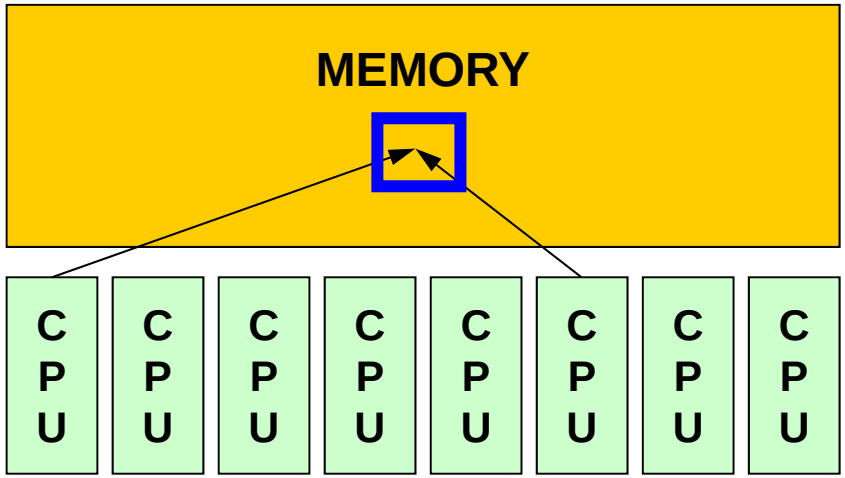
OpenMP/Direceives Matrix/Vector Products

```
#pragma omp parallel for private (i,VAL,j)
 for(i=0; i<N; i++) {
     VAL = D[i] * W[P][i];
     for(j=indexL[i]; j<indexL[i+1]; j++) {
         VAL += AL[j] * W[P][itemL[j]-1];
     }

     for(j=indexU[i]; j<indexU[i+1]; j++) {
         VAL += AU[j] * W[P][itemU[j]-1];
     }
     W[Q][i] = VAL;
 }
```

Features of OpenMP

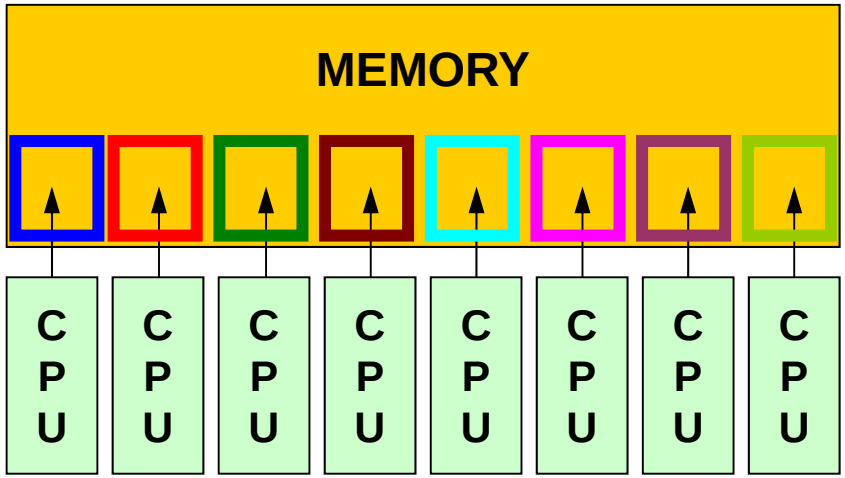
- Directives
  - Loops right after the directives are parallelized.
  - If the compiler does not support OpenMP, directives are considered as just comments.
- Nothing happen without explicit directives
  - Different from “automatic parallelization/vectorization”
  - Something wrong may happen by un-proper 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.)

- “for” loops with “#pragma omp parallel for”
- Global (Shared) Variables, Private Variables
  - Default: Global (Shared)
  - Dot Products: reduction

W[:][:], R, Z
global (shared)

```
RHO = 0.0;
#pragma omp parallel for private (i) reduction (+:RHO)
 for(i=0; i<N; i++) {
     RHO += W[R][i] * W[Z][i];
 }
```

FORTRAN & C

```
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
```

```
#include <omp.h>
...
{
  #pragma omp parallel for default(none) shared(n,x,y) private(i)

    for (i=0; i<n; i++)
        x[i] += y[i];
}
```

In this class ...

- There are many capabilities of OpenMP.
- In this class, only several functions are shown for parallelization of ICCG solver.

First things to be done

- 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, COMP, *$OMP
    - !$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)

```
#pragma omp directive_name [clause[[,] clause]…]
```

- “＼” for continuation lines
- Only lower case (except names of variables)

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

PARALLEL DO

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

```
#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

```
#pragma omp parllel for
 for(i=0; i<N; i++){
     B[i]= (A[i] + B[i]) * 0.50;
 }
```

- Default status of loop variables (“i” in this case) is private. Therefore, explicit declaration is not needed.

Example-1: REDUCTION

```
#pragma omp parallel default(private) reduction(+:A,B)
 for(i=0; i<N; i++){
     err= work(Alocal, Blocal);
     A= A + Alocal;
     B= B + Blocal;
 }
```

Functions in OpenMP

functions	description
int omp_get_num_threads (void)	Thread #
int omp_get_thread_num (void)	Thread ID
double omp_get_wtime (void)	Timer
void omp_set_num_threads (int num_threads) call omp_set_num_threads (num_threads)	Specifying Thread #

OpenMP for Dot Products

```
VAL= 0.0;
for(i=0; i<N; i++){
  VAL= VAL + W[R][i] * W[Z][i];
}
```

OpenMP for Dot Products

```
    VAL= 0.0;
    for(i=0; i<N; i++){
      VAL= VAL + W[R][i] * W[Z][i];
    }
```

```
    VAL= 0.0;
#pragma omp parallel for private (i) reduction(+:VAL)
    for(i=0; i<N; i++){
      VAL= VAL + W[R][i] * W[Z][i];
    }
```

Directives are just inserted.

OpenMP for Dot Products

```
      VAL= 0.0;
      for(i=0; i<N; i++){
        VAL= VAL + W[R][i] * W[Z][i];
      }
```

```
      VAL= 0.0;
#pragma omp parallel for private (i) reduction(+:VAL)
      for(i=0; i<N; i++){
        VAL= VAL + W[R][i] * W[Z][i];
      }
```

Directives are just inserted.

```
       VAL= 0.0;
#pragma omp parallel for private (i,ip)
 reduction(+:VAL)
      for(ip=0; ip<PEsmpTOT; ip++){
          for (i=INDEX[ip]; i<INDEX[ip+1]; i++){
             VAL= VAL + W[R][i] * W[Z][i];
          }
      }
```

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= 0.0;
#pragma omp parallel for private (i,ip)
 reduction(+:VAL)
    for(ip=0; ip<PEsmpTOT; ip++){
        for (i=INDEX[ip]; i<INDEX[ip+1]; i++){
           VAL= VAL + W[R][i] * W[Z][i];
        }
    }
```

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

Matrix-Vector Multiply

```
for(i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
        VAL += AL[j] * W[P][itemL[j]-1];
    }

    for(j=indexU[i]; j<indexU[i+1]; j++) {
        VAL += AU[j] * W[P][itemU[j]-1];
    }
    W[Q][i] = VAL;
}
```

Matrix-Vector Multiply

```
#pragma omp parallel for private(ip,i,VAL,j)
 for (ip=0; ip<PEsmpTOT; ip++){
      for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++){
          VAL = D[i] * W[P][i];
          for(j=indexL[i]; j<indexL[i+1]; j++) {
              VAL += AL[j] * W[P][itemL[j]-1];
          }

          for(j=indexU[i]; j<indexU[i+1]; j++) {
              VAL += AU[j] * W[P][itemU[j]-1];
          }
          W[Q][i] = VAL;
      }
 }
```

Matrix-Vector Multiply: Other Approach

This is rather better for GPU and (very) many-core architectures: simpler structure of loops

```
#pragma omp parallel for private(i,VAL,j)
 for(i=0; i<N; i++){
          VAL = D[i] * W[P][i];
          for(j=indexL[i]; j<indexL[i+1]; j++) {
              VAL += AL[j] * W[P][itemL[j]-1];
          }

          for(j=indexU[i]; j<indexU[i+1]; j++) {
              VAL += AU[j] * W[P][itemU[j]-1];
          }
          W[Q][i] = VAL;
 }
```

omp parallel (do)

- Each “omp parallel-omp end parallel” pair starts & stops threads: fork-join
- If you have many loops, these operations on threads could be overhead
- omp parallel + omp do/omp for

```
!$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 {
```

- OpenMP
- **Login to FX10 (separate file)** [1] [2]
- Parallel Version of the Code by OpenMP
- STREAM
- Using the Profiler of FX10
- Data Dependency

- OpenMP
- Login to FX10
- **Parallel Version of the Code by OpenMP**
- STREAM
- Using the Profiler of FX10
- Data Dependency

Target for Parallelization

- FVM code
- Preconditioned CG solver: PCG
  - Diagonal Scaling, Point Jacobi
- NO sample code.
- Please develop the parallel code by yourself

Preconditioned Conjugate Gradient Method（PCG）

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    solve [M]z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
     else
      βi-1= ρi-1/ρi-2
      p(i)= p(i-1) + βi-1 z(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

Solving the following equation:

$$\{z\} = [M]^{-1}\{r\}$$

“Approximate Inverse Matrix”

$$[M]^{-1} \approx [A]^{-1}, \quad [M] \approx [A]$$

Ultimate Preconditioning:
Inverse Matrix

$$[M]^{-1} = [A]^{-1}, \quad [M] = [A]$$

Diagonal Scaling: Simple but weak

$$[M]^{-1} = [D]^{-1}, \quad [M] = [D]$$

Diagonal Scaling, Point-Jacobi

$$[M] = \begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

- **solve $[M]z^{(i-1)} = r^{(i-1)}$** is very easy.
- Provides fast convergence for simple problems.

Files on Oakleaf-FX (1/2)

```
>$ cd

>$ cp /home/z30088/omp/omp-c.tar .
>$ cp /home/z30088/omp/omp-c2.tar .
>$ cp /home/z30088/omp/omp-f.tar .

>$ tar xvf omp-c.tar
>$ tar xvf omp-c2.tar
>$ tar xvf omp-f.tar

>$ cd multicore
```

Confirm Directories:

```
omp  stream
```

```
<$O-omp>, <$O-stream>
```

Files on Oakleaf-FX (2/2)

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

>$ make

>$ cd ../run
>$ pjsub go.sh
```

<$O-omp>/src/Makefile

NOT for parallel computing

```
CC          = fccpx           : Compiler
OPTFLAGS    = -Kfast          : Optimization
TARGET      = ../run/sol      : Exec File

.SUFFIXES:
.SUFFIXES: .o .c

.c.o:
        $(CC) -c $(CFLAGS) $(OPTFLAGS) $< -o $@

OBJS = input.o pointer_init.o ¥
       boundary_cell.o cell_metrics.o ¥
       rcm.o ...

HEADERS = ¥
        struct.h struct_ext.h ¥
        pcg.h ...

all: $(TARGET)
$(TARGET): $(OBJS)
        $(CC) $(CFLAGS) $(OPTFLAGS) -o $@ $(OBJS) $(LDFLAGS)

$(OBJS): $(HEADERS)

clean:
        rm -f *.o $(TARGET) *.log *~ *.lst
```

Running Job

- Batch Jobs
  - Only batch jobs are allowed.
  - Interactive executions of jobs are not allowed.
- How to run
  - writing job script
  - submitting job
  - checking job status
  - checking results
- Utilization of computational resources
  - 1-node (16 cores) is occupied by each job.
  - Your node is not shared by other jobs.

Job Script

- **<$O-omp>/run/go.sh**
- Scheduling + Shell Script

```
#!/bin/sh
#PJM -L “node=1“                Number of Nodes
#PJM -L “elapse=00:10:00“       Computation Time
#PJM -L “rscgrp=lecture9“       Name of “QUEUE”
#PJM -g “gt89“                  Group Name (Wallet)
#PJM -j
#PJM -o “test.lst“              Standard Output

./sol                           Execs
```

Available QUEUE's

- Following 2 queues are available.
- 1 Tofu (12 nodes) can be used
  - **`lecture`**
    - 12 nodes (192 cores), 15 min., **valid until the end of October 2015**
    - Shared by all "educational" users
  - **`lecture9`**
    - 12 nodes (192 cores), 15 min., active during class time
    - **More jobs (compared to `lecture`) can be processed up on availability.**
    - **Just during classes (except June 26(F))**

Submitting & Checking Jobs

• Submitting Jobs	`pjsub SCRIPT NAME`
• Checking status of jobs	`pjstat`
• Deleting/aborting	`pjdel JOB ID`
• Checking status of queues	`pjstat --rsc`
• Detailed info. of queues	`pjstat --rsc –x`
• Number of running jobs	`pjstat --rsc –b`
• Limitation of submission	`pjstat --limit`

```
[z30088@oakleaf-fx-6 S2-ref]$ pjstat

Oakleaf-FX scheduled stop time: 2012/09/28(Fri) 09:00:00 (Remain: 31days 20:01:46)

JOB_ID     JOB_NAME   STATUS  PROJECT  RSCGROUP  START_DATE        ELAPSE          TOKEN NODE:COORD
334730     go.sh      RUNNING gt61     lecture   08/27 12:58:08    00:00:05          0.0 1
```

solve_PCG (1/3)

```
for(i=0; i<N; i++) {
    W[DD][i]= 1.e0/D[i];
}

...


for(L=0; L<(*ITR); L++) {

/*******************
 * {z} = [Minv]{r} *
 *******************/
    for(i=0; i<N; i++) {
        W[Z][i] = W[R][i]*W[DD][i];
    }


/*****************
 * RHO = {r}{z} *
 *****************/
    RHO = 0.0;
    for(i=0; i<N; i++) {
        RHO += W[R][i] * W[Z][i];
    }
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i\ = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

solve_PCG (2/3)

```
/**********************************
 * {p}  = {z} if      ITER=0    *
 * BETA = RHO / RHO1  otherwise *
 **********************************/

if(L == 0) {
   for(i=0; i<N; i++) {
       W[P][i] = W[Z][i];
   }
   } else {
   BETA = RHO / RHO1;
   for(i=0; i<N; i++) {
       W[P][i] = W[Z][i] + BETA * W[P][i];
   }
}

/*****************
 * {q} = [A]{p} *
 *****************/

for(i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
        VAL += AL[j] * W[P][itemL[j]-1];
    }
    for(j=indexU[i]; j<indexU[i+1]; j++) {
        VAL += AU[j] * W[P][itemU[j]-1];
    }
    W[Q][i] = VAL;
}
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

solve_PCG (3/3)

```
/*************************
 * ALPHA = RHO / {p}{q} *
 *************************/
C1 = 0.0;
for(i=0; i<N; i++) {
    C1 += W[P][i] * W[Q][i];
}
ALPHA = RHO / C1;

/*****************************
 * {x} = {x} + ALPHA * {p} *
 * {r} = {r} - ALPHA * {q} *
 *****************************/
for(i=0; i<N; i++) {
    X[i]    += ALPHA * W[P][i];
    W[R][i] -= ALPHA * W[Q][i];
}

DNRM2 = 0.0;
for(i=0; i<N; i++) {
    DNRM2 += W[R][i]*W[R][i];
}

ERR = sqrt(DNRM2/BNRM2);
if((L+1)%100 ==1) {
   fprintf(stderr, "%5d%16.6e¥n", L+1, ERR);
}
if(ERR < EPS) {
   *IER = 0; goto N900;
  } else {
   RHO1 = RHO;
  }
}
*IER = 1;
```

r= b-[A]x
DNRM2=$|r|^2$
BNRM2=$|b|^2$

ERR= |r|/|b|

Compute $r^{(0)}$= b-[A]$x^{(0)}$
for i= 1, 2, …
  solve [M]$z^{(i-1)}$= $r^{(i-1)}$
  ρ_{i-1}= $r^{(i-1)}$ $z^{(i-1)}$
  if i=1
    $p^{(1)}$= $z^{(0)}$
  else
    β_{i-1}= ρ_{i-1}/ρ_{i-2}
    $p^{(i)}$= $z^{(i-1)}$ + β_{i-1} $p^{(i)}$
  endif
  $q^{(i)}$= [A]$p^{(i)}$
  α_i = $\rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}$= $x^{(i-1)}$ + $\alpha_i p^{(i)}$
  $r^{(i)}$= $r^{(i-1)}$ - $\alpha_i q^{(i)}$
  check convergence |r|
end

Parallelization by OpenMP

- Focusing on “solver_PCG.c” (solve_PCG)
- Just insert OpenMP directives

```
>$ cd <$O-omp>
>$ cd ex
(modify files)

>$ make
>$ cd ../run
>$ pjsub g.sh
```

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:10:00"
#PJM -L "rscgrp=lecture9"
#PJM -g "gt89"
#PJM -j
#PJM -o "100_08_002.lst"

export OMP_NUM_THREADS=8      1-16
./sol
```

<$O-omp>/ex/Makefile

parallel computing by OpenMP

```
CC         = fccpx           : Compiler
OPTFLAGS   = -Kfast,openmp   : Optimization + OpenMP
TARGET     = ../run/sol      : Exec File

.SUFFIXES:
.SUFFIXES: .o .c

.c.o:
        $(CC) -c $(CFLAGS) $(OPTFLAGS) $< -o $@

OBJS = input.o pointer_init.o ¥
       boundary_cell.o cell_metrics.o ¥
       rcm.o ...

HEADERS = ¥
        struct.h struct_ext.h ¥
        pcg.h ...

all: $(TARGET)
$(TARGET): $(OBJS)
        $(CC) $(CFLAGS) $(OPTFLAGS) -o $@ $(OBJS) $(LDFLAGS)

$(OBJS): $(HEADERS)

clean:
        rm -f *.o $(TARGET) *.log *~ *.lst
```

solve_PCG (1/5)

parallel computing by OpenMP

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include <omp.h>

#include "solver_PCG.h"

extern int
solve_PCG (int N, int NL, int NU, int *indexL, int *itemL, int *indexU, int *itemU,
                double *D, double *B, double *X, double *AL, double *AU,
                double EPS, int *ITR, int *IER, int *N2)
{
        double **W;
        double VAL, BNRM2, WVAL, SW, RHO, BETA, RHO1, C1, DNRM2, ALPHA, ERR;
        double Stime, Etime;
        int i, j, ic, ip, L, ip1, N3;
        int R = 0;
        int Z = 1;
        int Q = 1;
        int P = 2;
        int DD = 3;
```

solve_PCG (2/5)

```
#pragma omp parallel for private (i)
        for(i=0; i<N; i++) {
                X[i] = 0.0;
                W[1][i] = 0.0;
                W[2][i] = 0.0;
                W[3][i] = 0.0;
        }

#pragma omp parallel for private (i,VAL,j)
        for(i=0; i<N; i++) {
                VAL = D[i] * X[i];
                for(j=indexL[i]; j<indexL[i+1]; j++) {
                        VAL += AL[j] * X[itemL[j]-1];
                }
                for(j=indexU[i]; j<indexU[i+1]; j++) {
                        VAL += AU[j] * X[itemU[j]-1];
                }
                W[R][i] = B[i] - VAL;
        }

BNRM2 = 0.0;
#pragma omp parallel for private (i) reduction (+:BNRM2)
        for(i=0; i<N; i++) {
          BNRM2 += B[i]*B[i];
        }

#pragma omp parallel for private (i)
        for(i=0; i<N; i++) {
          W[DD][i]= 1.e0/D[i];
        }
```

Compute $r^{(0)}= b-[A]x^{(0)}$

for i= 1, 2, …
- solve $[M]z^{(i-1)}= r^{(i-1)}$
- $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
- if i=1
  - $p^{(1)}= z^{(0)}$
- else
  - $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
  - $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
- endif
- $q^{(i)}= [A]p^{(i)}$
- $\alpha_i\ = \rho_{i-1}/p^{(i)}q^{(i)}$
- $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
- $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
- check convergence |r|

end

solve_PCG (3/5)

```
*ITR = N;

Stime = omp_get_wtime();

for(L=0; L<(*ITR); L++) {

#pragma omp parallel for private (i)
        for(i=0; i<N; i++) {
            W[Z][i] = W[R][i]*W[DD][i];
        }

RHO = 0.0;
#pragma omp parallel for private (i) reduction(+:RHO)
        for(i=0; i<N; i++) {
            RHO += W[R][i] * W[Z][i];
        }

if(L == 0) {
#pragma omp parallel for private (i)
        for(i=0; i<N; i++) {
            W[P][i] = W[Z][i];
          }
} else {
        BETA = RHO / RHO1;
#pragma omp parallel for private (i)
        for(i=0; i<N; i++) {
            W[P][i] = W[Z][i] + BETA * W[P][i];
        }
}
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

solve_PCG (4/5)

```
#pragma omp parallel for private (i,VAL,j)
        for(i=0; i<N; i++) {
            VAL = D[i] * W[P][i];
            for(j=indexL[i]; j<indexL[i+1]; j++) {
                VAL += AL[j] * W[P][itemL[j]-1];
            }
            for(j=indexU[i]; j<indexU[i+1]; j++) {
                VAL += AU[j] * W[P][itemU[j]-1];
            }
            W[Q][i] = VAL;
        }

C1 = 0.0;
#pragma omp parallel for private (i) reduction(+:C1)
        for(i=0; i<N; i++) {
            C1 += W[P][i] * W[Q][i];
        }
        ALPHA = RHO / C1;

#pragma omp parallel for private (i)
        for(i=0; i<N; i++) {
            X[i]    += ALPHA * W[P][i];
            W[R][i] -= ALPHA * W[Q][i];
        }

DNRM2 = 0.0;
#pragma omp parallel for private (i) reduction(+:DNRM2)
        for(i=0; i<N; i++) {
            DNRM2 += W[R][i]*W[R][i];
        }

        ERR = sqrt(DNRM2/BNRM2);
...
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

solve_PCG (5/5)

```
        Stime = omp_get_wtime();

for(L=0; L<(*ITR); L++) {
...
        if(ERR < EPS) {
           *IER = 0;
           goto N900;
        } else {
           RHO1 = RHO;
        }
}
        *IER = 1;
N900:
        Etime = omp_get_wtime();

        fprintf(stderr, "%5d%16.6e¥n", L+1, ERR);
        fprintf(stderr, "%16.6e sec. (solver)¥n", Etime - Stime);

        *ITR = L;
        free(W);
        return 0;
}
```

Elapsed Time= Etime - Stime

Etime-Stime

NX=NY=NZ=100, 10^6 DOF

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:10:00"
#PJM -L "rscgrp=lecture9"
#PJM -g "gt89"
#PJM -j
#PJM -o "100_08_002.lst"

export OMP_NUM_THREADS=M      M=1,4 8,16
./sol
```

M	sec.	Speed-Up
1	39.4	1.00
4	10.1	3.90
8	5.29	7.45
16	3.37	11.7

Exercises

- Develop your own program by inserting OpenMP directives !

- Effect of problem size (NX, NY, NZ)
- Effect of Thread # (OMP_NUM_THREADS: 1-16)

- OpenMP
- Login to FX10
- Parallel Version of the Code by OpenMP
- **STREAM**
- Using the Profiler of FX10
- Report S1
- Data Dependency

Why less than 16x ?

- Memory Contention
- Performance of memory per each thread decreases if number of threads on each node increases
- Sparse Matrix Solver: Memory-Bound
  - Effect of this decreasing is more significant
- Problem size is not so larger

Sparse/Dense Matrices

```
for(i=0;i<N;i++){
    Y[i] = Diag[i] * X[i];
    for(k=Index[i];k<Index[i+1];k++){
        Y[i] += AMat[k]*X[Item[k]];
    }
}
```

```
for(j=0;j<N;i++){
  Y[j]= 0.0;
  for (i=0;i<N;i++){
    Y[j]+= A[j][i]*X[i];
  }
}
```

- “X” in RHS
  - Dense: continuous on memory, easy to utilize cache
  - Sparse: continuity is not assured, difficult to utilize cache
    - more “memory-bound”

GeoFEM Benchmark

ICCG in FEM for Solid Mechanics

	SR11K/J2	SR16K/M1	T2K	FX10	京
Core #/Node	16	32	16	16	8
Peak Performance (GFLOPS)	147.2	980.5	147.2	236.5	128.0
STREAM Triad (GB/s)	101.0	264.2	20.0	64.7	43.3
B/F	0.686	0.269	0.136	0.274	0.338
GeoFEM (GFLOPS)	19.0	72.7	4.69	16.0	11.0
% to Peak	12.9	7.41	3.18	6.77	8.59
LLC/core (MB)	18.0	4.00	2.00	0.75	0.75

Sparse Linear Solver: Memory-Bound

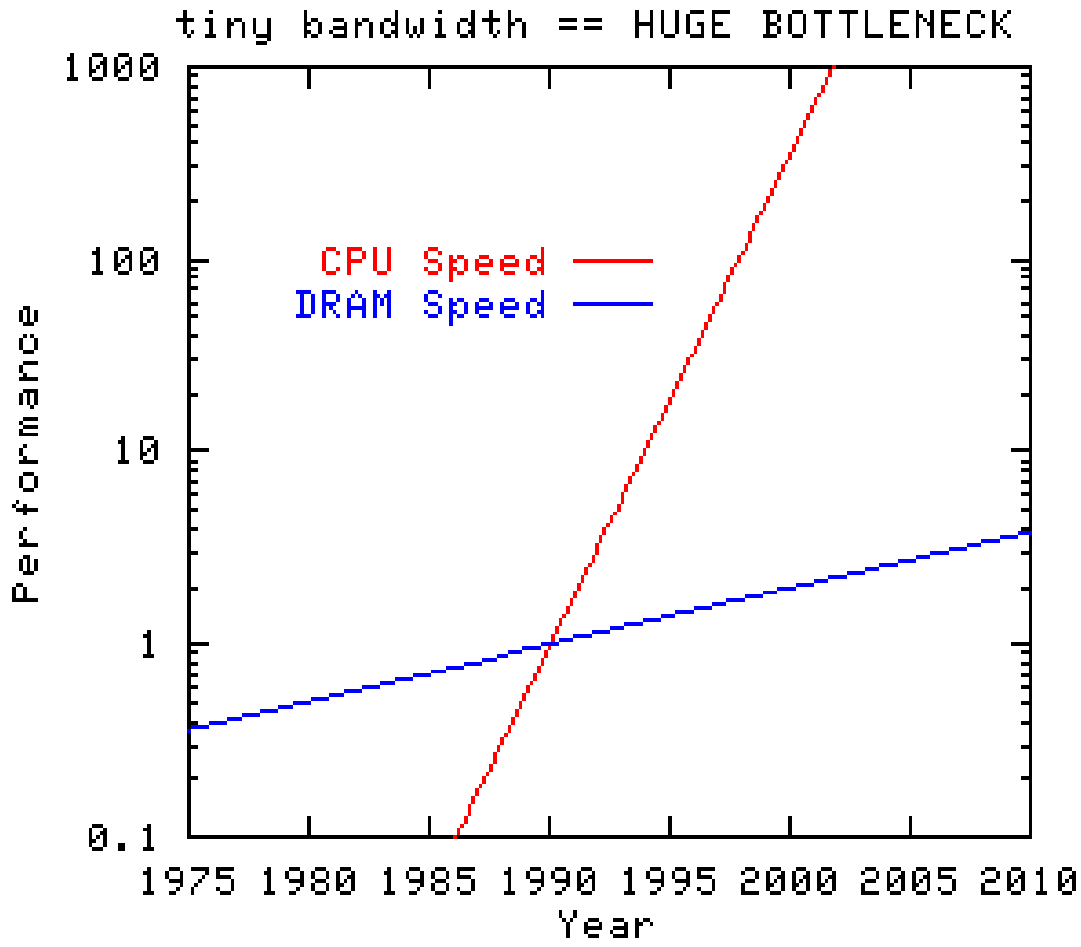
STREAM benchmark

http://www.cs.virginia.edu/stream/

- Benchmarks for Memory Bandwidth
  - Copy: c(i)= a(i)
  - Scale: c(i)= s*b(i)
  - Add: c(i)= a(i) + b(i)
  - Triad: c(i)= a(i) + s*b(i)

```
-------------------------------------------------
 Double precision appears to have 16 digits of accuracy
 Assuming 8 bytes per DOUBLE PRECISION word
-------------------------------------------------
 Number of processors =           16
 Array size =    2000000
 Offset     =          0
 The total memory requirement is     732.4 MB
(     45.8MB/task)
 You are running each test  10 times
 --
 The *best* time for each test is used
 *EXCLUDING* the first and last iterations
 -----------------------------------------------------
 -----------------------------------------------------
Function     Rate (MB/s)  Avg time   Min time  Max time
Copy:      18334.1898      0.0280      0.0279      0.0280
Scale:     18035.1690      0.0284      0.0284      0.0285
Add:       18649.4455      0.0412      0.0412      0.0413
Triad:     19603.8455      0.0394      0.0392      0.0398
```

Gap between performance of CPU and Memory



http://www.cs.virginia.edu/stream/

OpenMP version of STREAM

```
>$ cd <$O-stream>
>$ pjsub go.sh
```

- http://www.cs.virginia.edu/stream/
- C, Fortran, MPI, OpenMP etc.

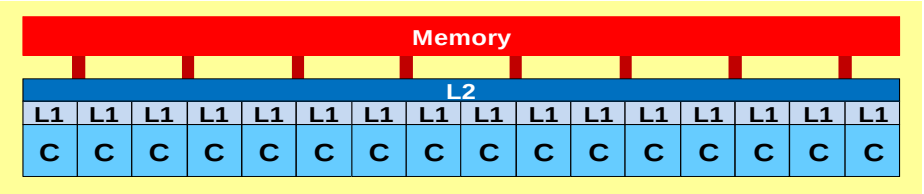
go.sh

```
#!/bin/sh
#PJM -L "rscgrp=tutorial"
#PJM -L "node=1"
#PJM -L "elapse=10:00"
#PJM -j

export PATH=…
export LD_LIBRARY_PATH=…
export PARALLEL=16
export OMP_NUM_THREADS=16        Number of threads (1-16)

./stream.out > 16-01.lst 2>&1    Name of output file
```

Results of Triad

<$O-stream>/stream/*.lst

Peak is 85.3 GB/sec., 75%



Thread #	MB/sec.	Speed-up
1	8606.14	1.00
2	16918.81	1.97
4	34170.72	3.97
8	59505.92	6.91
16	64714.32	7.52

Exercises

- Running the code
- Try various number of threads (1-16)
- MPI-version and Single PE version are available
  - Fortran, C
  - Web-site of STREAM

- OpenMP
- Login to FX10
- Parallel Version of the Code by OpenMP
- STREAM
- **Using the Profiler of FX10**
- Data Dependency

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 avilable
  - Please use “-Nsrc”
  - Displayed on screen

Current version of C/C++ compiler can produce list of messages

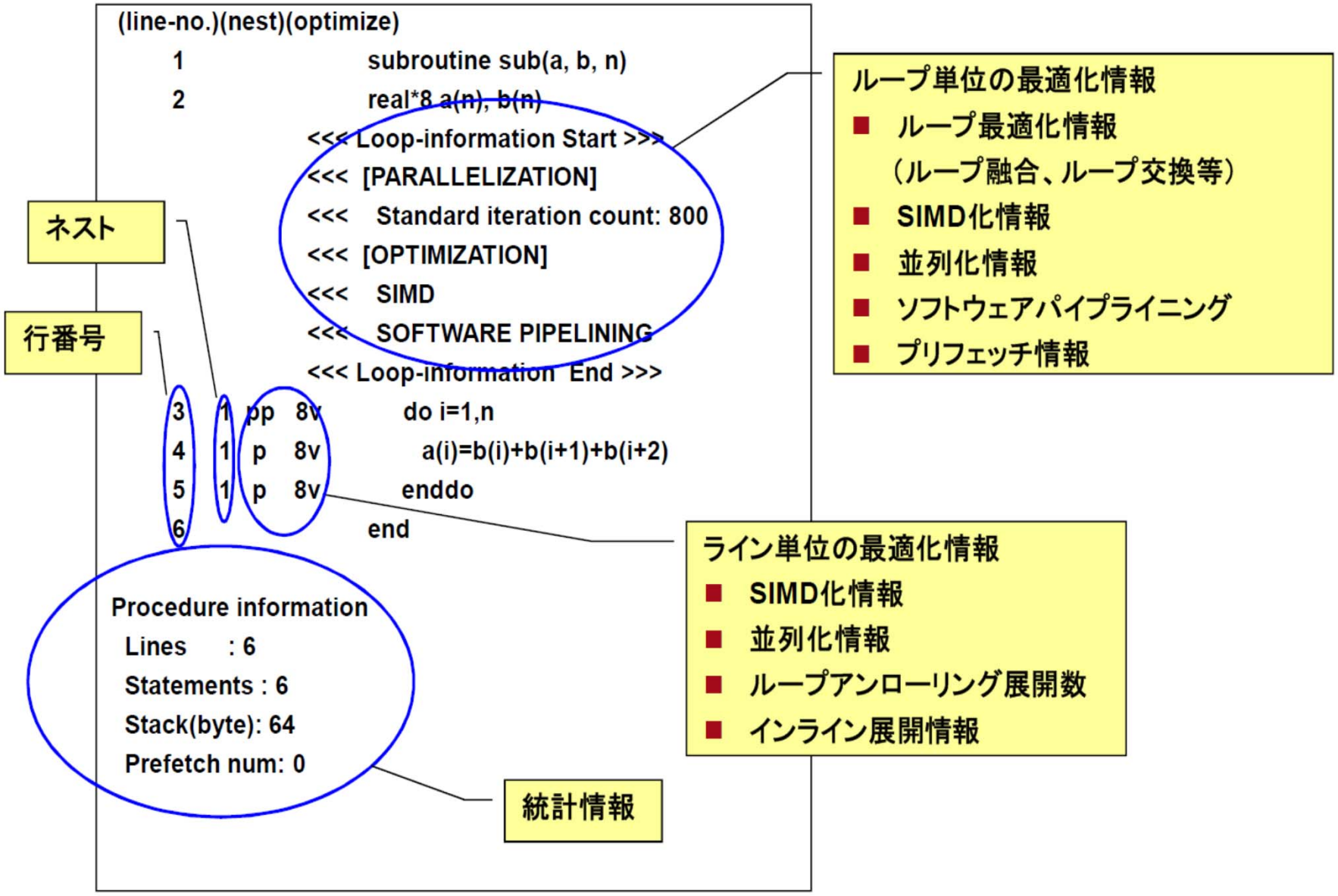
Fortran/C/C++

-Nlst=p 標準の最適化情報 (デフォルト)
-Nlst=t 詳細な最適化情報

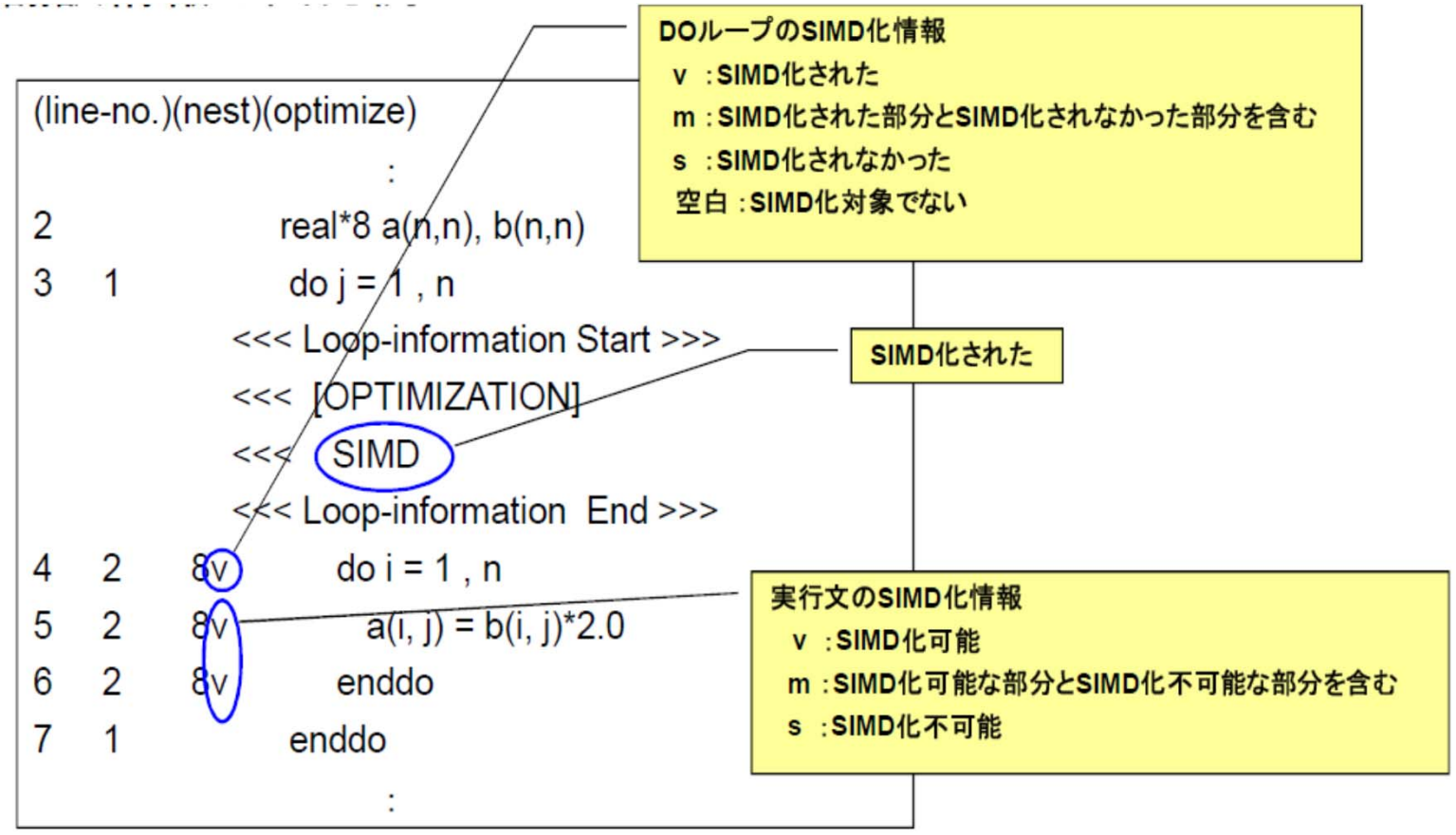
Fortran ONLY

-Nlst=a 名前の属性情報
-Nlst=d 派生型の構成情報
-Nlst=i 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
-Nlst=m 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
-Nlst=x 名前および文番号の相互参照情報

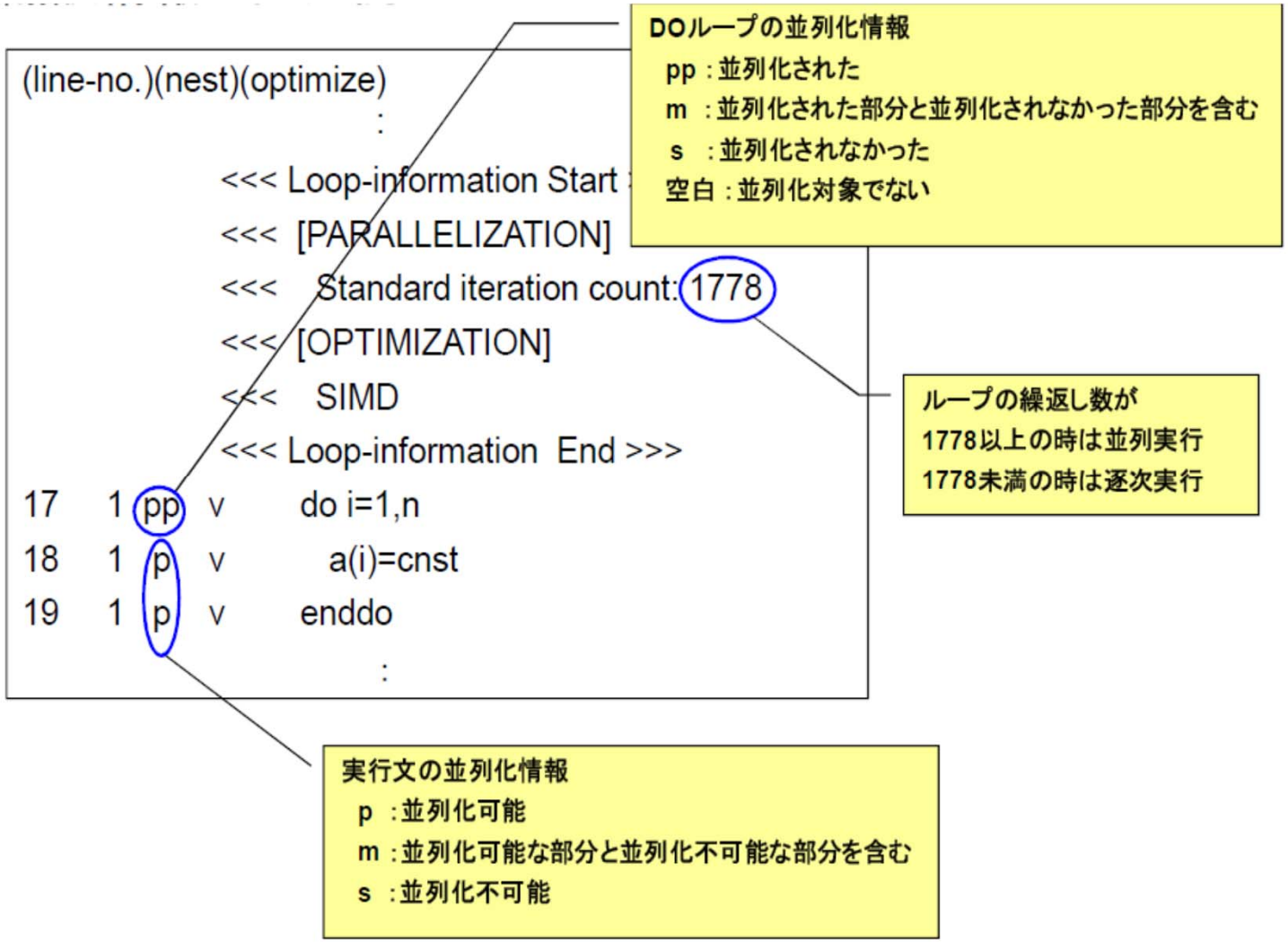
Info in *.lst


(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 1 pp 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 end
Procedure information
Lines : 6
Statements : 6
Stack(byte): 64
Prefetch num: 0
ネスト
行番号
統計情報
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ニング
■ 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

SIMD Information

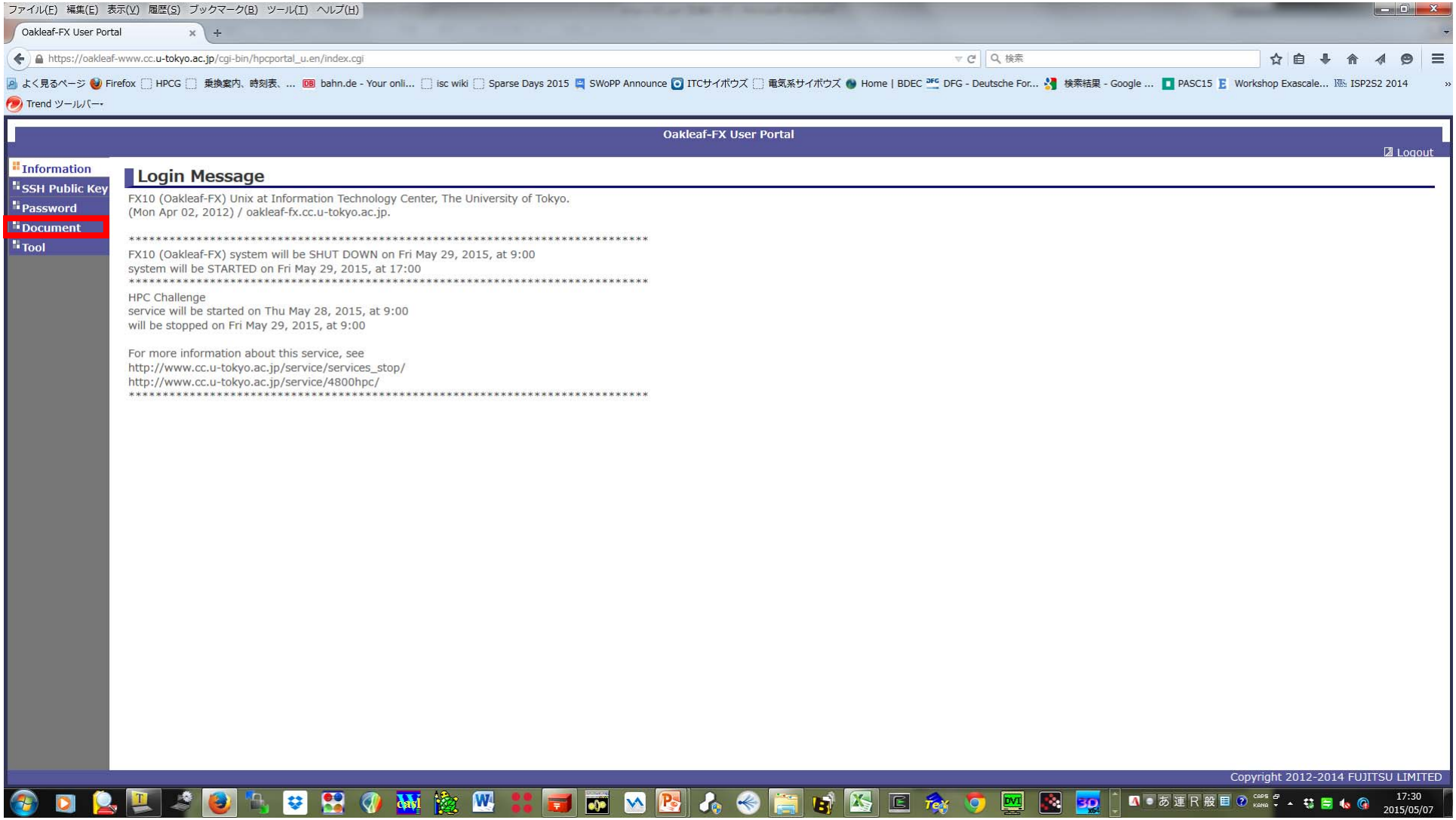


Automatic Parallelization



Profiles (1/3)

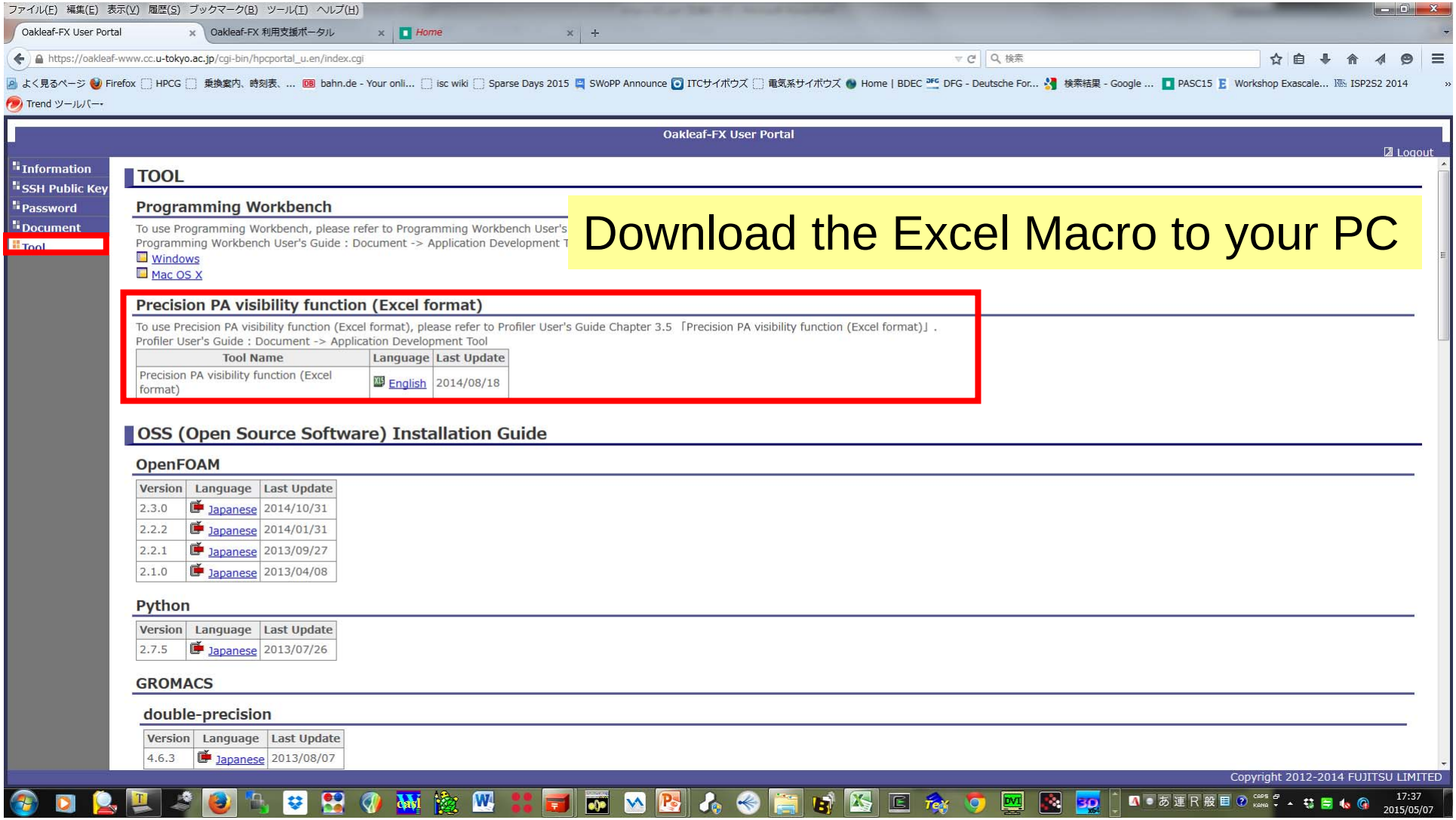
- https://oakleaf-www.cc.u-tokyo.ac.jp/cgi-bin/hpcportal.en/index.cgi
- https://oakleaf-www.cc.u-tokyo.ac.jp/cgi-bin/hpcportal.ja/index.cgi



Profiles (2/3)


Oakleaf-FX User Portal
Logout
Information
SSH Public Key
Password
Document
Tool
XPFortran
Document Name | Language | Last Update
XPFortran User's Guide | English | 2014/08/18
Application Development Tool
User's Guide
Document Name | Language | Last Update
Runtime Information Output Function User's Guide | English | 2014/08/18
Programming Workbench User's Guide | English | 2014/08/18
Profiler User's Guide | English | 2014/08/18
Debugger User's Guide | English | 2012/07/27
Rank Map Automatic Tuning Tools User's Guide | English | 2014/05/02
プロファイラ使用手引書
Numerical Libraries
Numerical Libraries User's Guide
Document Name | Language | Last Update
Programmer's Guide for Usage of Mathematical Libraries | English | 2014/08/18
Additional functions in Mathematical Library [Japanese Only] | Japanese | 2012/06/07
SSL II
Document Name | Language | Last Update
SSL II User's Guide | English | 2014/08/18
SSL II Extended Capabilities User's Guide | English | 2014/08/18
SSL II Extended Capabilities User's Guide II | English | 2014/08/18
SSL II Thread-Parallel Capabilities User's Guide | English | 2014/08/18
C-SSL II
Copyright 2012-2014 FUJITSU LIMITED

Profiles (3/3)


Oakleaf-FX User Portal
Logout
Information
SSH Public Key
Password
Document
Tool
TOOL
Programming Workbench
Windows
Mac OS X
Download the Excel Macro to your PC
Precision PA visibility function (Excel format)
To use Precision PA visibility function (Excel format), please refer to Profiler User's Guide Chapter 3.5 「Precision PA visibility function (Excel format)」.
Profiler User's Guide : Document -> Application Development Tool
Tool Name | Language | Last Update
Precision PA visibility function (Excel format) | English | 2014/08/18
OSS (Open Source Software) Installation Guide
OpenFOAM
Version | Language | Last Update
2.3.0 | Japanese | 2014/10/31
2.2.2 | Japanese | 2014/01/31
2.2.1 | Japanese | 2013/09/27
2.1.0 | Japanese | 2013/04/08
Python
Version | Language | Last Update
2.7.5 | Japanese | 2013/07/26
GROMACS
double-precision
Version | Language | Last Update
4.6.3 | Japanese | 2013/08/07
Copyright 2012-2014 FUJITSU LIMITED

3.5 精密PA可視化機能 (Excel) (1/4) (Precision PA Visibility Function)

Inserting Call's, Compile & Run

```
      call start_collection ( "CG")
      Stime= omp_get_wtime()
...
      Etime= omp_get_wtime()
      call stop_collection ( "CG")
```

```
      start_collection ( "CG");
      Stime= omp_get_wtime();
...
      Etime= omp_get_wtime();
      stop_collection ( "CG");
```

- Detailed profile between "start_collection" and "stop_collection" can be obtained.
- You can specify multiple segments in the program

3.5 精密PA可視化機能 (Excel) (2/4) (Precision PA Visibility Function)

Collecting Performance Data: 7X Exec's Directories: pa1~pa7, -Hpa=1~7

```
>$ cd <$O-omp>
>$ cd ex
(modify files)

>$ make
>$ cd ../run

>$ pjsub gp.sh
```

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:15:00"
#PJM -L "rscgrp=lecture9"
#PJM -g "gt89"
#PJM -j
#PJM -o "a.lst"

export OMP_NUM_THREADS=16
fapp -C -d pa1 -Hpa=1  ./sol
fapp -C -d pa2 -Hpa=2  ./sol
fapp -C -d pa3 -Hpa=3  ./sol
fapp -C -d pa4 -Hpa=4  ./sol
fapp -C -d pa5 -Hpa=5  ./sol
fapp -C -d pa6 -Hpa=6  ./sol
fapp -C -d pa7 -Hpa=7  ./sol
```

3.5 精密PA可視化機能 (Excel) (3/4) (Precision PA Visibility Function)

Performance Analysis: Transformation

```
>$ cd <$O-omp>/run

>$ ./file.sh
>$ ls -l ~/output_prof_*.csv
```

```
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
```

3.5 精密PA可視化機能 (Excel) (4/4) (Precision PA Visibility Function)

Copy the files & Excel

```
>$ cd <$cur>

copy FSDT_CPUPA.xlsm/FSDT_CPUPA_ENG.xlsm to <$cur>

>$ scp t89***@oakleaf-fx.cc.u-tokyo.ac.jp:~/output_prof_* .

start Excel
```

- MFLOPS
- Memory Throughput (GB/s): Compare with STREAM
- Instruction／命令

- OpenMP
- Login to FX10
- Parallel Version of the Code by OpenMP
- STREAM
- Using the Profiler of FX10
- **Data Dependency**

Parallelize ICCG Method

- Dot Product: OK
- DAXPY: OK
- Matrix-Vector Multiply: OK
- Preconditioning

How about the preconditioning ? Point Jacobi is easy: but slow

```
  for (i=0; i<N; i++){
    W[Z][i]= W[R][i]*W[DD][i];
  }
```

```
#omp pragma parallel for private(i)
  for (i=0; i<N; i++){
    W[Z][i]= W[R][i]*W[DD][i];
  }
```

```
#omp pragma parallel for private(i,ip)
  for (ip=0; ip<PEsmpTOT; ip++){
  for (i=INDEX[ip]; i<INDEX[ip+1]; i++)
    W[Z][i]= W[R][i]*W[DD][i];
  }
  }
```

```
64*64*64
METHOD= 1
1     6.543963E+00
101   1.748392E-05
146   9.731945E-09

real    0m14.662s
```

```
METHOD= 3
1     6.299987E+00
101   1.298539E+00
201   2.725948E-02
301   3.664216E-05
401   2.146428E-08
413   9.621688E-09

real    0m19.660s
```

How about the preconditioning ?

IC Factorization

```
for(i=0; i<N; i++) {
  VAL = D[i];
  for(j=indexL[i]; j<indexL[i+1]; j++) {
    VAL -= AL[j]*AL[j]*W[DD][itemL[j]-1];
  }
  W[DD][i] = 1.0 / VAL;
}
```

Forward Substitution

```
for(i=0; i<N; i++) {
  WVAL = W[Z][i];
  for(j=indexL[i]; j<indexL[i+1]; j++) {
    WVAL -= AL[j] * W[Z][itemL[j]-1];
   }
   W[Z][i] = WVAL * W[DD][i];
}
```

Data Dependency

Conflict of reading from/writing to memory
Difficult to be parallelized

IC Factorization

```
for(i=0; i<N; i++) {
  VAL = D[i];
  for(j=indexL[i]; j<indexL[i+1]; j++) {
    VAL -= AL[j]*AL[j]*W[DD][itemL[j]-1];
  }
  W[DD][i]= 1.0 / VAL;
}
```

Forward Substitution

```
for(i=0; i<N; i++) {
  WVAL = W[Z][i];
  for(j=indexL[i]; j<indexL[i+1]; j++) {
    WVAL -= AL[j] * W[Z][itemL[j]-1];
   }
   W[Z][i]= WVAL * W[DD][i];
}
```

Forward Substitution
Four Thread Parallel Operation

“icel” starts at 0

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

“itemL” starts at 1

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

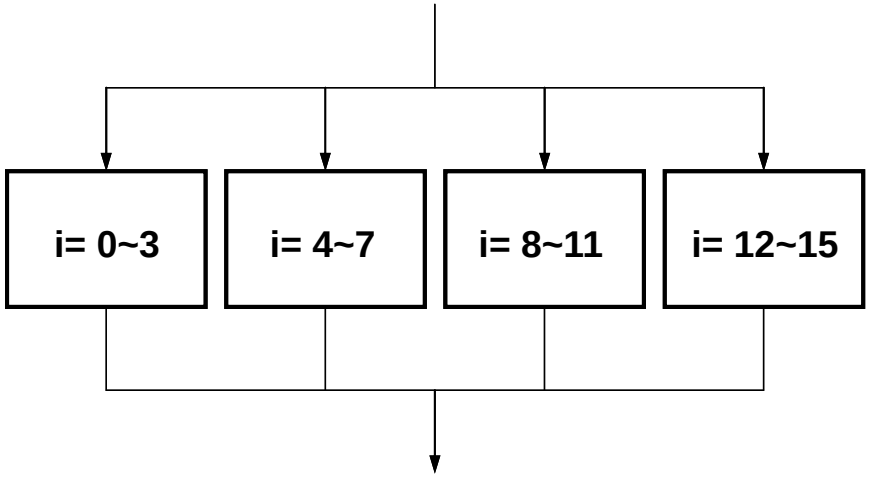
```
for(i=0; i<N; i++) {
  VAL = D[i];
  for(j=indexL[i]; j<indexL[i+1]; j++) {
    VAL -= AL[j]*AL[j]*W[DD][itemL[j]-1];
  }
  W[DD][i]= 1.0 / VAL;
}
```

Forward Substitution
Four Thread Parallel Operation

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

```
#pragma omp parallel private (ip,i,k,VAL)
for(ip=1; ip<4; ip++) {
for(i=INDEX[ip]; i<INDEX[ip+1];i++){
  VAL = D[i];
  for(j=indexL[i]; j<indexL[i+1]; j++) {
    VAL -= AL[j]*AL[j]*W[DD][itemL[j]-1];
  }
  W[DD][i]= 1.0 / VAL;
}
}
```

```
INDEX[0]= 0
INDEX[1]= 4
INDEX[2]= 8
INDEX[3]=12
INDEX[4]=16
```



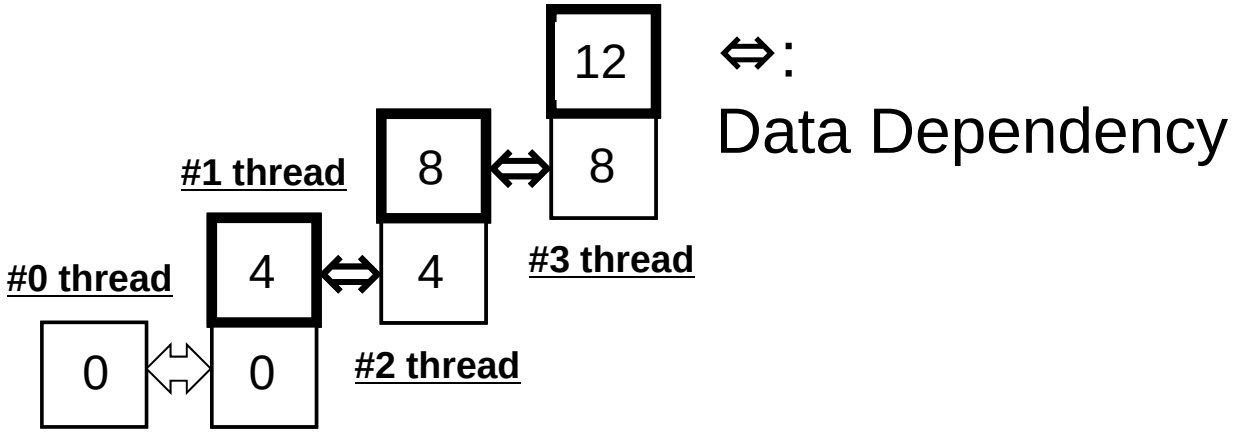
These four threads are executed simultaneously.

Data Dependency: Conflict of reading from/writing to memory

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

```
#pragma omp parallel private (ip,i,k,VAL)
for(ip=1; ip<4; ip++) {
for(i=INDEX[ip]; i<INDEX[ip+1];i++){
  VAL = D[i];
  for(j=indexL[i]; j<indexL[i+1]; j++) {
    VAL -= AL[j]*AL[j]*W[DD][itemL[j]-1];
  }
  W[DD][i]= 1.0 / VAL;
}
}
```

```
INDEX[0]= 0
INDEX[1]= 4
INDEX[2]= 8
INDEX[3]=12
INDEX[4]=16
```



Parallelize ICCG Method

- Dot Product: OK
- DAXPY: OK
- Matrix-Vector Multiply: OK
- Preconditioning: Something special needed !
  - Just inserting OpenMP directive is not enough