

Introduction to Parallel Programming for Multicore/Manycore Clusters

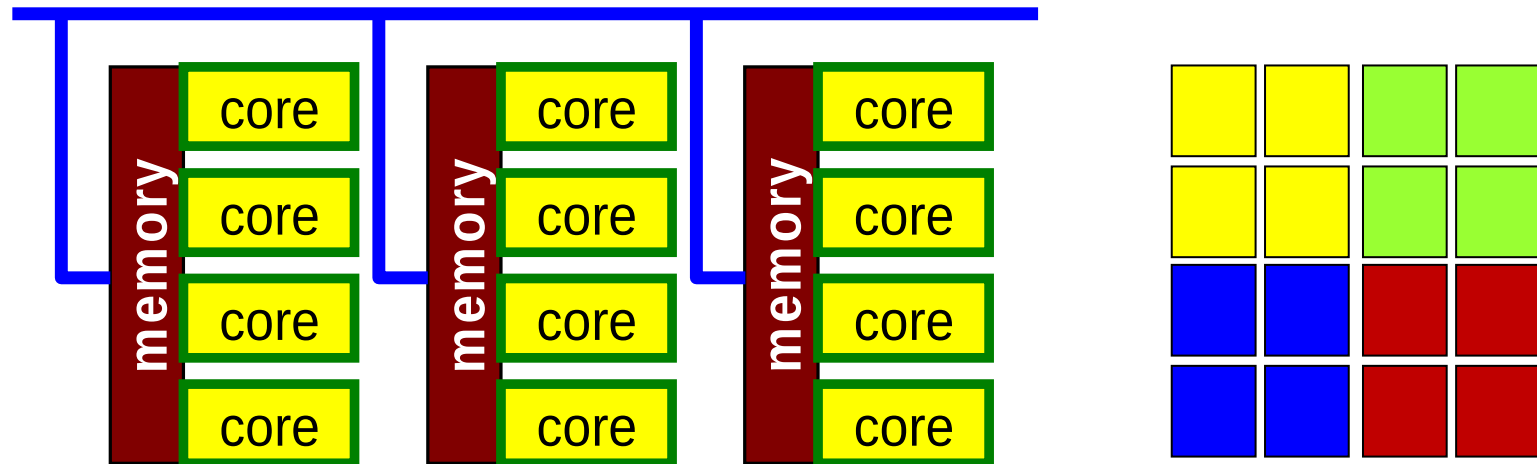
Part II: Introduction to OpenMP

Kengo Nakajima
Information Technology Center
The University of Tokyo

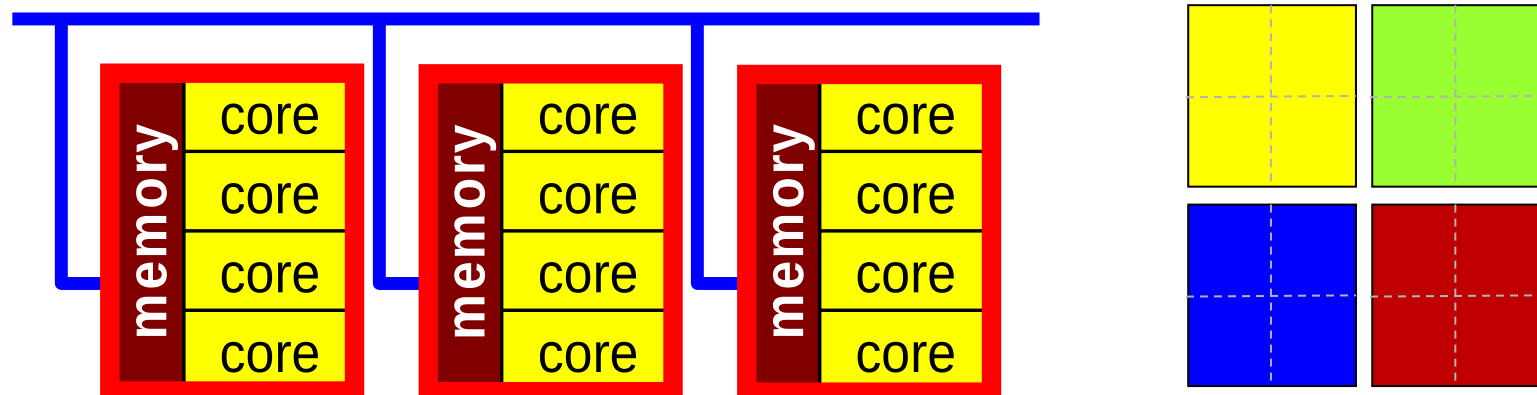
- OpenMP
- Login to OBCX
- Parallel Version of the Code by OpenMP
- STREAM
- Data Dependency

Flat MPI vs. Hybrid

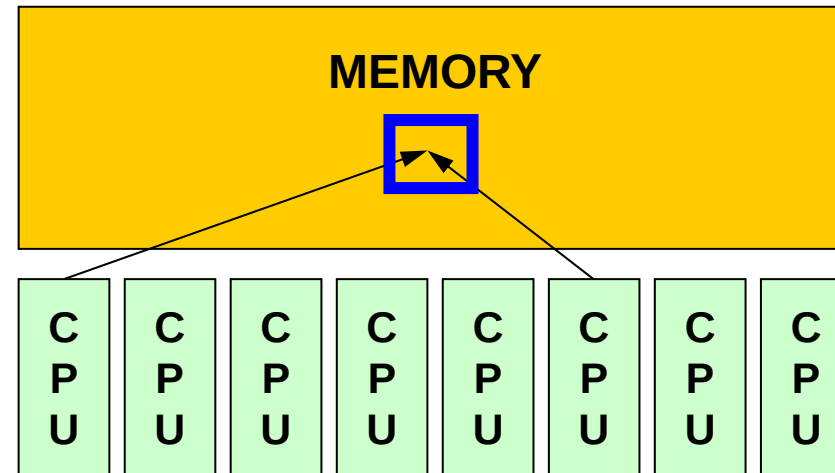
Flat-MPI: Each Core -> Independent



Hybrid: Hierarchical Structure



SMP



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

What is OpenMP ? (1/2)

<http://www.openmp.org>

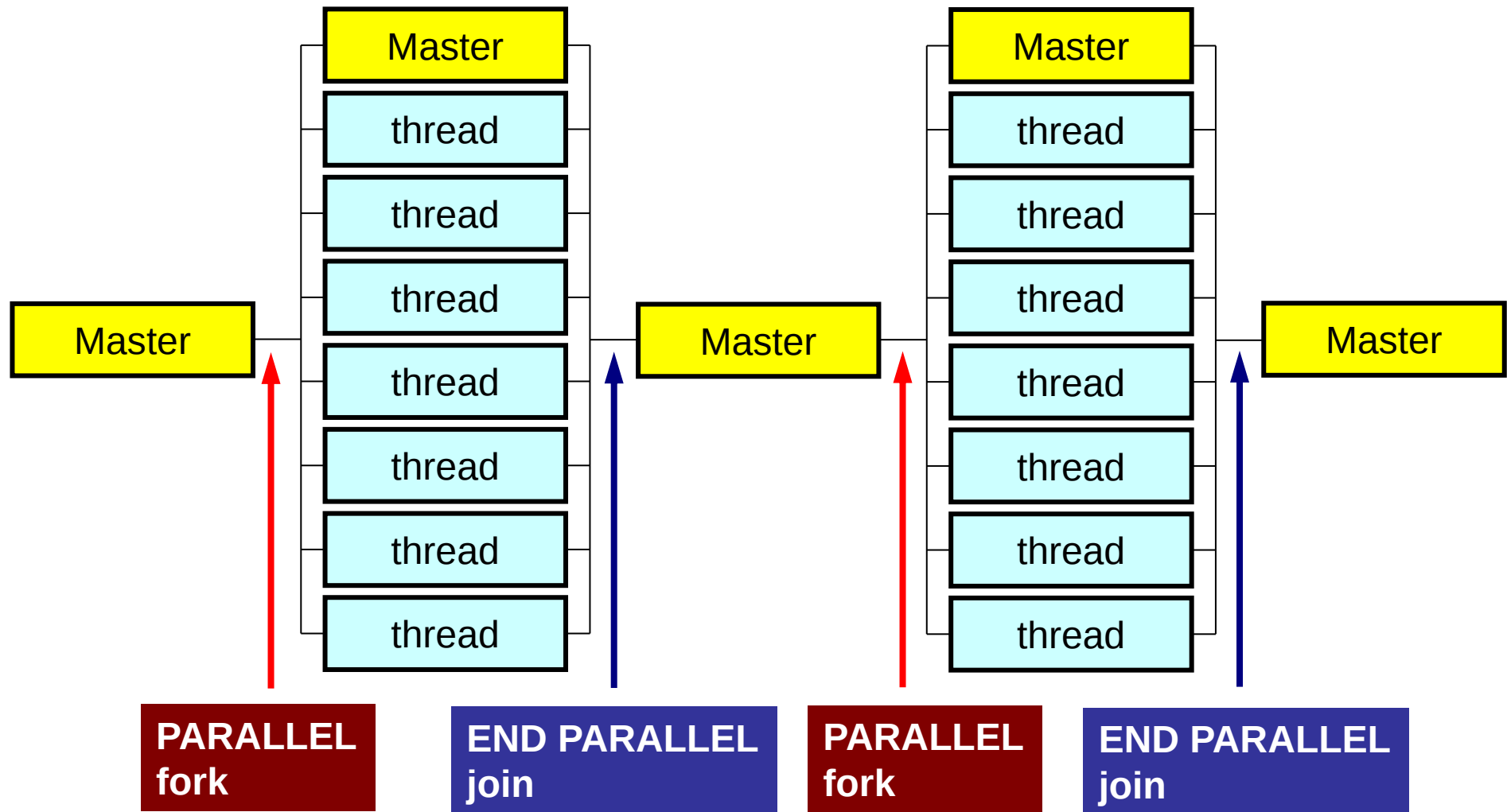
- An API (Application Programming Interface) for multi-platform shared-memory parallel programming in C/C++ and Fortran
 - Current version: 4.X (5.0 is already announced)
 - GPU, Accelerators: close to OpenACC
- Background
 - Merger of Cray and SGI in 1996
 - Separated later, ... but both are now merged into HPE
 - ASCI project (US-DOE (Dept. of Energy)) started in 1995
 - Accelerated Strategic Computing Initiative (ASCI) -> Advanced Simulation and Computing Program (ASC)
 - The goal of ASCI is to simulate the results of new weapons designs as well as the effects of aging on existing and new designs, all in the absence of additional data from underground nuclear tests.
 - Development of Supercomputers & Software/Applications
 - SMP Clusters: Intel ASCI Red, IBM Power (Blue, White, Purple)/Blue Gene, SGI
 - Common API for SMP Clusters needed

What is OpenMP ? (2/2)

<http://www.openmp.org>

- C/C++ version and Fortran version have been separately developed until ver.2.5.
- Fork-Join Parallel Execution Model (Next Page)
 - Directives: Parallel, End Parallel
 - Serial Execution: Master Thread
 - Parallel Execution: Master Thread/Thread Team
- 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 ?

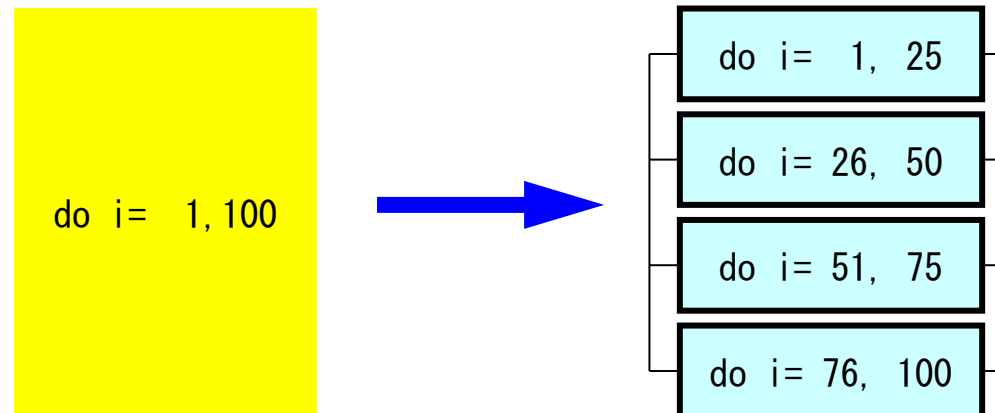
- `bash(.bashrc)`

- `export OMP_NUM_THREADS=8`

- `csh(.cshrc)`

- `setenv OMP_NUM_THREADS 8`

- **OMP_NUM_THREADS=4**



Information about OpenMP

- OpenMP Architecture Review Board (ARB)
 - <http://www.openmp.org>
 - Spec. of OpenMP is available
- 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)
- 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

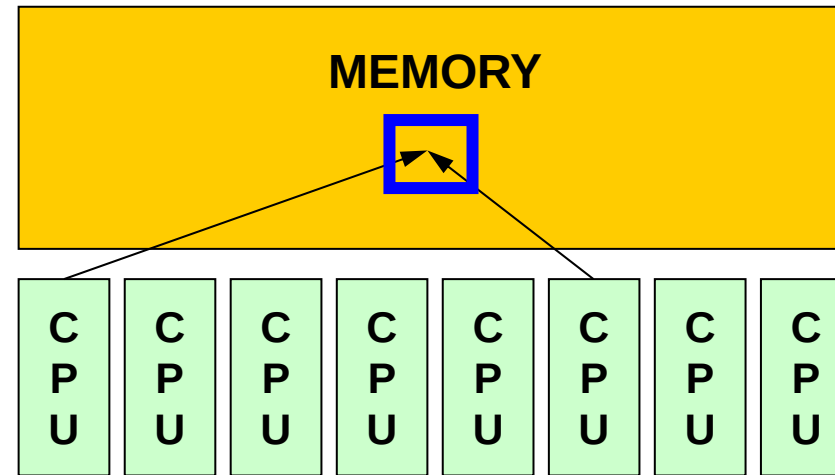
```
#pragma omp parallel for private (i, VAL, j)
for (i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for (j=indexL[i]; j<indexL[i+1]; j++) {
        VAL += AL[j] * W[P][itemL[j]-1];
    }

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

Features of OpenMP

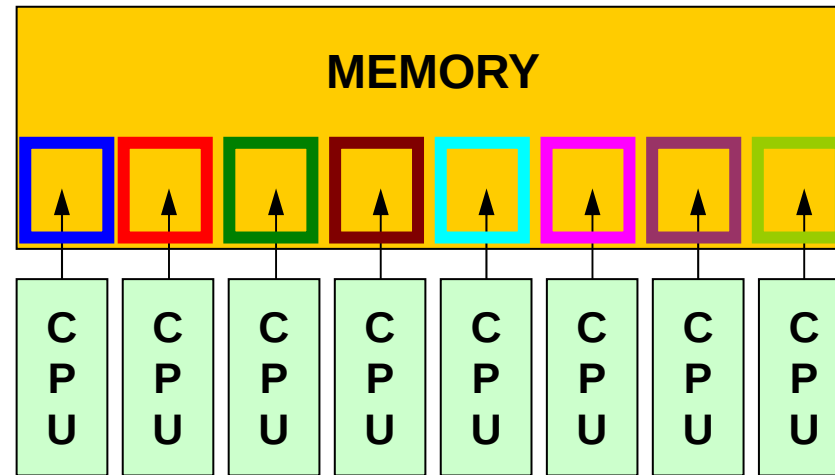
- Directives
 - Loops right after the directives are parallelized.
 - If the compiler does not support OpenMP, directives are considered as just comments.
- Nothing happen without explicit directives
 - Different from “automatic parallelization/vectorization”
 - Something wrong may happen by un-proper 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, any such case does not happen by reordering etc.
 - In OpenMP, users are responsible for such issues (e.g. proper data configuration, reordering etc.)
- Data Dependency
- Performance per core reduces as number of used cores (thread #) increases (Memory Saturation)

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)

```
RH0 = 0.0;  
#pragma omp parallel for private (i) reduction (+:RH0)  
for (i=0; i<N; i++) {  
    RH0 += 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, 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)

```
#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/for

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

```
#pragma omp 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 parallel 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
<code>int omp_get_num_threads (void)</code>	Thread #
<code>int omp_get_thread_num (void)</code>	Thread ID
<code>double omp_get_wtime (void)</code>	Timer
<code>void omp_set_num_threads (int num_threads)</code> call <code>omp_set_num_threads (num_threads)</code>	Specifying Thread #

OpenMP for Dot Products

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

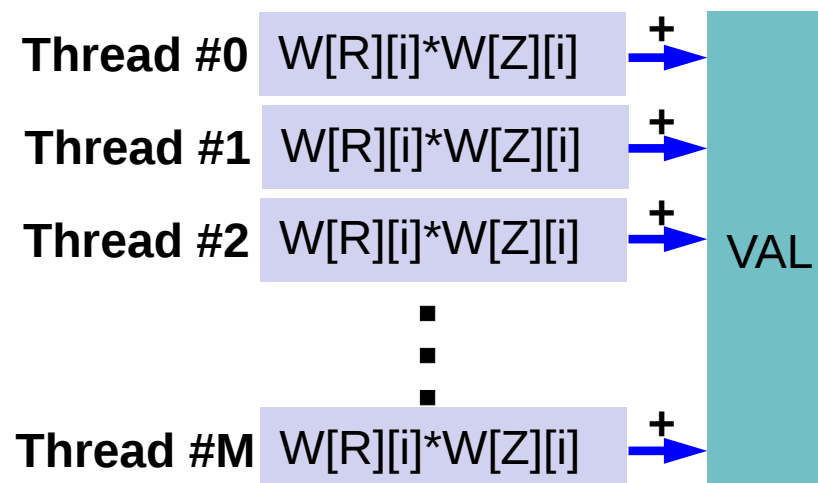
OpenMP for Dot Products

```
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

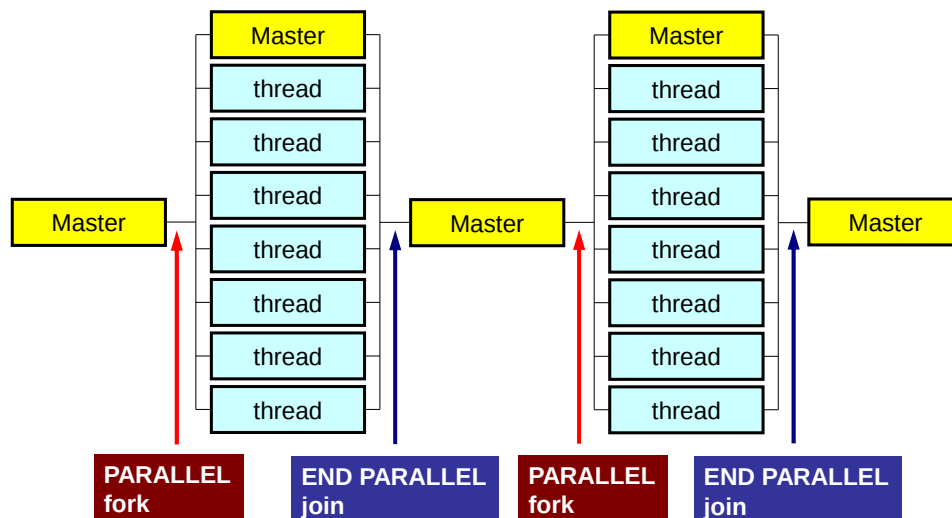
This is rather better for GPU and (very) many-core architectures: simpler structure of loops

```
#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)

- “omp parallel-omp end parallel” = “fork-join”
- If you have many loops, these “fork-join’s” cause operations
- **omp parallel + omp do/omp for**



```
#pragma omp parallel ...
```

```
#pragma omp for {
```

```
...
```

```
#pragma omp for {
```

```
!$omp parallel ...
```

```
!$omp do
```

```
    do i= 1, N
```

```
...
```

```
!$omp do
```

```
    do i= 1, N
```

```
...
```

```
!$omp end parallel required
```

- OpenMP
- **Login to OBCX (separate file)**
- Parallel Version of the Code by OpenMP
- STREAM
- Data Dependency

- OpenMP
- Login to OBCX
- **Parallel Version of the Code by OpenMP**
- STREAM
- Data Dependency

Target for Parallelization

- FVM code
- Preconditioned CG solver: PCG
 - Diagonal Scaling, Point Jacobi (METHOD=3)

Preconditioned Conjugate Gradient Method (PCG)

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
    solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} \cdot z^{(i-1)}$ 
    if  $i = 1$ 
         $p^{(1)} = z^{(0)}$ 
    else
         $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
         $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
    endif
     $q^{(i)} = [A]p^{(i)}$ 
     $\alpha_i = \rho_{i-1} / p^{(i)} \cdot q^{(i)}$ 
     $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
     $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
    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 $[M] \mathbf{z}^{(i-1)} = \mathbf{r}^{(i-1)}$** is very easy.
- Provides fast convergence for simple problems.

Original: 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
         $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
         $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
    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

```

Reciprocal numbers (倒数) of diagonal components are stored in W[DD][i]. Computational cost for division is usually expensive.

Although it was said (division):(+, -, *) is 10:1 before, the difference is much smaller now. Generally, multiplying is still faster than division.

Original: solve_PCG (2/3)

```

/*****
* {p} = {z} if ITER=0 *
* BETA = RHO / RH01 otherwise *
*****/

if(L == 0) {
    for(i=0; i<N; i++) {
        W[P][i] = W[Z][i];
    }
    else {
        BETA = RHO / RH01;
        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 $i = 1, 2, \dots$
 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
 $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
 $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
 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

Original: 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  $[M]z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    if i=1
         $p^{(1)} = z^{(0)}$ 
    else
         $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
         $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
    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

```

Files on OBCX

```
>$ cd /work/gt69/t69XXX
```

```
>$ cp /work/gt69/z30088/omp/omp-c.tar .
```

Please do not forget this “.”

```
>$ tar xvf omp-c.tar
```

```
>$ cd multicore
```

```
>$ ls  
omp stream
```

Hereafter: <\$O-omp>, <\$O-stream>

```
>$ cd omp/src20
```

```
>$ make
```

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

(modify go1.sh, go2.sh)

```
>$ pjsub go1.sh
```

```
>$ pjsub go2.sh
```

If you have errors in compiling, please add “-no-multibyte-chars” at “OPTFLAGS” of <\$O-omp>/src20/Makefile

OPTFLAGS= -O3 (...) -no-multibyte-chars

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, RH01, 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] = 1.0/D[i];
}

```

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

```

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) = z(i-1) + βi-1 p(i-1)
    endif
    q(i) = [A] p(i)
    αi = ρi-1 / p(i) q(i)
    x(i) = x(i-1) + αi p(i)
    r(i) = r(i-1) - αi 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 $i = 1, 2, \dots$

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

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

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
         $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
         $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
    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 {  
        RH01 = RH0;  
    }  
}  
*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

<\$O-omp>/src20/Makefile

parallel computing by OpenMP

```

CC          = icc
OPTFLAGS= -O3 -axCORE-AVX512 -align -qopenmp -ipo
TARGET      = ../run/sol20

.SUFFIXES:
.SUFFIXES: .o .c

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

OBJS = input.o pointer_init.o boundary_cell.o cell_metrics.o ¥
      poi_gen.o solver_PCG.o outucd.o allocate.o main.o

HEADERS = struct.h struct_ext.h pcg.h pcg_ext.h input.h pointer_init.h ¥
          boundary_cell.h cell_metrics.h poi_gen.h solver_PCG.h ¥
          allocate.h outucd.h
all: $(TARGET)
$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) $(OPTFLAGS) -o $@ $(OBJS) $(LDFLAGS)

$(OBJS): $(HEADERS)

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

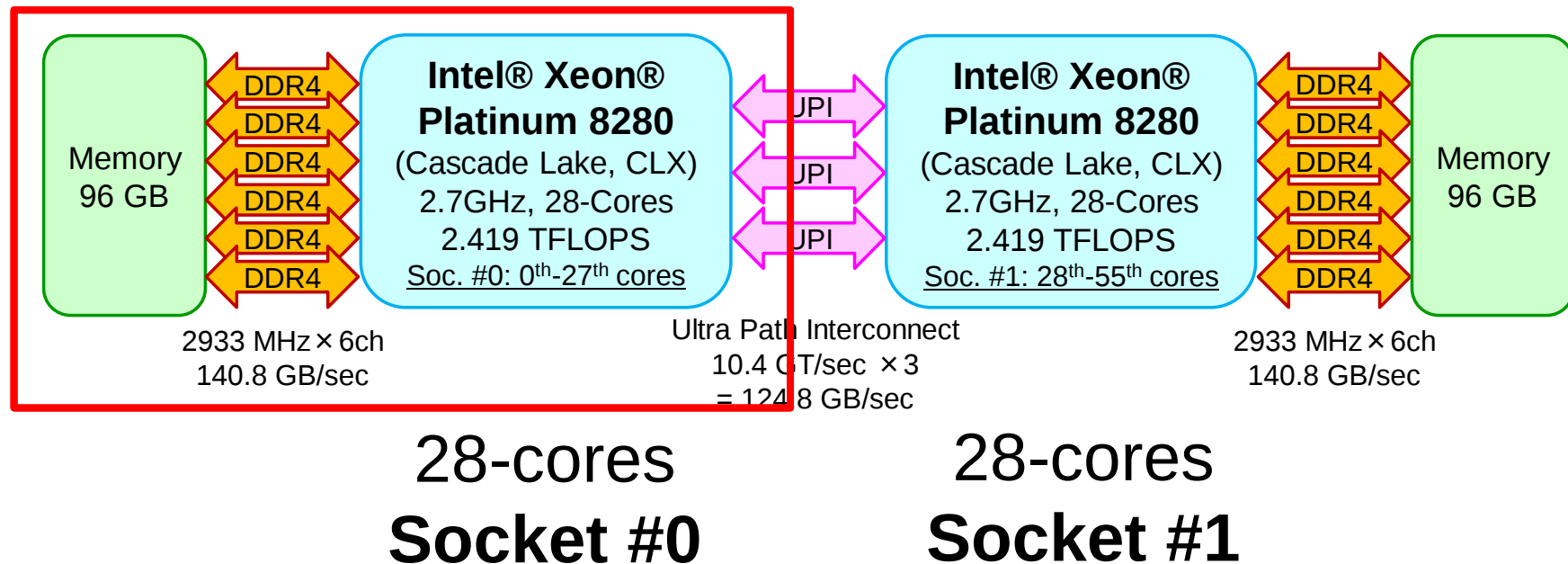
```

If you have errors in compiling, please add “-no-multibyte-chars” at “OPTFLAGS”
OPTFLAGS= -O3 (...) -no-multibyte-chars

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 (56 cores) is occupied by each job.
 - Your node is not shared by other jobs.

1 Node of OBCX consists of 2 CPU's (Sockets)



**Only a single socket
(Socket #0) will be
used in this class !**

Job Script (1/2) go1.sh

- `/work/gt69/t69XXX/multicore/omp/run/go1.sh`
- Scheduling + Shell Script

```
#!/bin/sh
#PJM -N "test1"           Job Name (not required)
#PJM -L rscgrp=lecture9    Name of Queue (Resource Grp.)
#PJM -L node=1            Node # (=1)
#PJM --omp thread=24       Thread # (=1-56)
#PJM -L elapse=00:15:00   Computation Time
#PJM -g gt69              Group Name (Wallet)
#PJM -j
#PJM -e err               Standard Error
#PJM -o test1.lst         Standard Output

export KMP_AFFINITY=granularity=fine,compact
./sol20                    Execution of the Program
```

```
export KMP_AFFINITY=granularity=fine,compact
All of 1-28 threads are on Socket #0
```

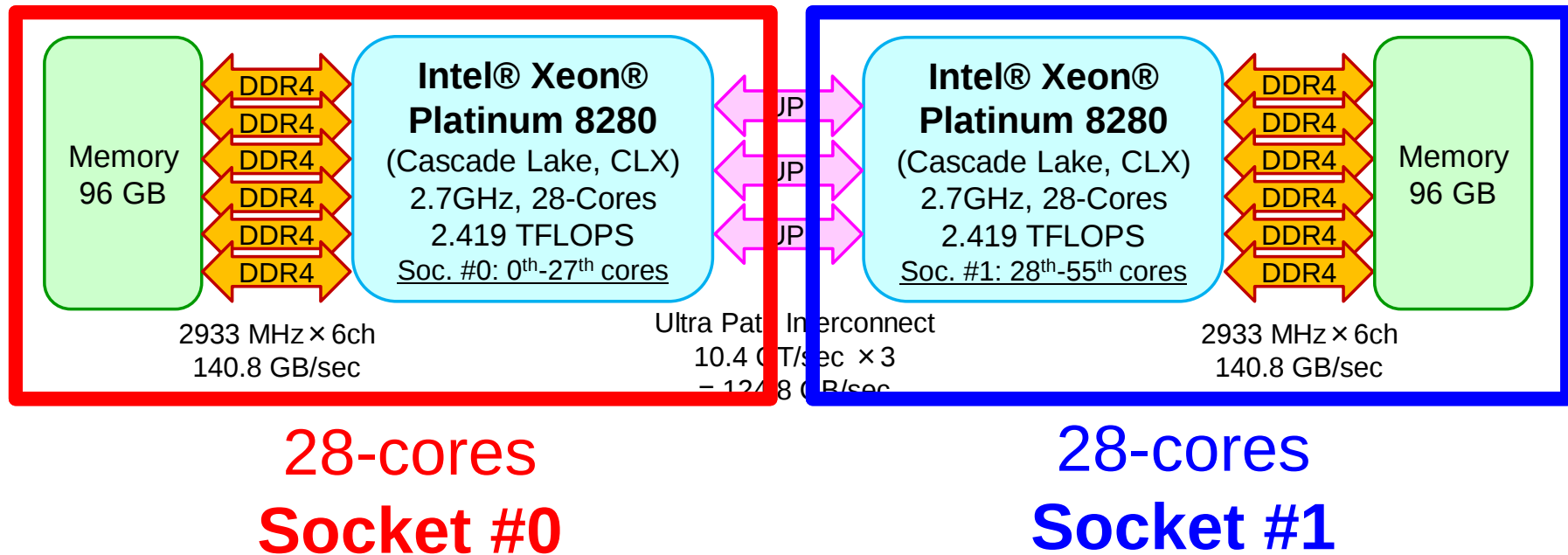
Job Script (2/2) go2.sh

- `/work/gt69/t69XXX/multicore/omp/run/go2.sh`
- Scheduling + Shell Script

<code>#!/bin/sh</code>	
<code>#PJM -N "test2"</code>	Job Name (not required)
<code>#PJM -L rscgrp=lecture9</code>	Name of Queue (Resource Grp.)
<code>#PJM -L node=1</code>	Node # (=1)
<code>#PJM --omp thread=24</code>	Thread # (=1-56)
<code>#PJM -L elapse=00:15:00</code>	Computation Time
<code>#PJM -g gt69</code>	Group Name (Wallet)
<code>#PJM -j</code>	
<code>#PJM -e err</code>	Standard Error
<code>#PJM -o test1.1st</code>	Standard Output
 <code>./sol20</code>	 Execution of the Program

Cores are randomly selected from Socket #0 & #1

1 Node of OBCX consists of 2 CPU's (Sockets)



go1.sh: cores on the Socket 0 are used

export KMP_AFFINITY=granularity=fine,compact

go2.sh: cores on the Socket 0 & Socket 1 are randomly used: May be faster than go1.sh

Running Jobs

```
>$ cd /work/gt69/t69XYZ  
>$ cd multicore/omp/run  
>$ pjsub gol.sh  
  
>$ cat test1.lst
```

INPUT.DAT

128	128	128		NX	NY	NZ
1.00e-0	1.00e-00	1.00e-00		DX	DY	DZ
1.0e-08				EPS	ICCG	

Available QUEUE's

- Following 2 queues are available.
- 8 nodes can be used
 - **lecture**
 - 8 nodes (448 cores), 15 min., valid until the end of Oct. 2021
 - Shared by all “educational” users
 - **lecture9**
 - 8 nodes (448 cores), 15 min., active during class time
 - More jobs (compared to **lecture**) can be processed up on availability.

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 -a`
- History of Submission `pjstat -H`
- Limitation of submission `pjstat --limit`

```
[t69XYZ@obcx04 run]$ pjsub go1.sh
```

```
[INFO] PJM 0000 pjsub Job 292019 submitted.
```

```
[t69XYZ@obcx04 run]$ pjsub go2.sh
```

```
[INFO] PJM 0000 pjsub Job 292020 submitted.
```

```
[t69XYZ@obcx04 run]$ pjstat
```

```
Oakbridge-CX scheduled stop time: 2020/04/24(Fri) 09:00:00 (Remain: 3days 23:09:15)
```

JOB_ID	JOB_NAME	STATUS	PROJECT	RSCGROUP	START_DATE	ELAPSE	TOKEN	NODE
292019	test1	RUNNING	gt69	lecture9	04/20 09:50:42<	00:00:02	-	1
292020	test2	QUEUED	gt69	lecture9	--/-- --:--:--	00:00:00	-	1

```
[t69XYZ@obcx04 run]$ pjstat
```

```
Oakbridge-CX scheduled stop time: 2020/04/24(Fri) 09:00:00 (Remain: 3days 23:09:12)
```

JOB_ID	JOB_NAME	STATUS	PROJECT	RSCGROUP	START_DATE	ELAPSE	TOKEN	NODE
292019	test1	RUNNING	gt69	lecture9	04/20 09:50:42<	00:00:06	-	1
292020	test2	RUNNING	gt69	lecture9	04/20 09:50:46<	00:00:02	-	1

```
[t69XYZ@obcx04 run]$ pjdel 292020
```

```
[INFO] PJM 0100 pjdel Job 292020 canceled.
```

```
[t69XYZ@obcx04 run]$ pjstat
```

```
Oakbridge-CX scheduled stop time: 2020/04/24(Fri) 09:00:00 (Remain: 3days 23:09:04)
```

JOB_ID	JOB_NAME	STATUS	PROJECT	RSCGROUP	START_DATE	ELAPSE	TOKEN	NODE
292019	test1	RUNNING	gt69	lecture9	04/20 09:50:42<	00:00:14	-	1

```
[t69XYZ@obcx04 run]$ pjstat
```

```
Oakbridge-CX scheduled stop time: 2020/04/24(Fri) 09:00:00 (Remain: 3days 23:07:14)
```

```
No unfinished job found.
```

```
[t69XYZ@obcx04 ~]$ pjstat --rsc
```

RSCGRP	STATUS	NODE
lecture	[ENABLE, START]	32
lecture1	[DISABLE, STOP]	64

```
[t69XYZ@obcx04 ~]$ pjstat --rsc -x
```

RSCGRP	STATUS	MIN_NODE	MAX_NODE	MAX_ELAPSE	REMAIN_ELAPSE	MEM (GB)	PROJECT
lecture	[ENABLE, START]	1	8	00:15:00	00:15:00	168	gt00
lecture1	[DISABLE, STOP]	1	8	00:15:00	--:--:--	168	gt00

```
[t69XYZ@obcx04 ~]$ pjstat -a
```

Oakbridge-CX scheduled stop time: 2020/04/24 (Fri) 09:00:00 (Remain: 3days 23:19:29)

JOB_ID	JOB_NAME	STATUS	PROJECT	RSCGROUP	START_DATE	ELAPSE	TOKEN	NODE
284147	*****	RUNNING	*****	-	04/19 12:58:20	--:--:--	-	-
284149	*****	RUNNING	*****	-	04/19 11:50:18	--:--:--	-	-
284159	*****	RUNNING	*****	-	04/19 19:16:10	--:--:--	-	-
289904	*****	RUNNING	*****	small	04/18 19:59:41	37:40:50	-	2
289909	*****	RUNNING	*****	small	04/19 01:02:58	32:37:33	-	2

(...)

```
[t69XYZ@obcx04 ~]$ pjstat -H
```

Oakbridge-CX scheduled stop time: 2020/04/24 (Fri) 09:00:00 (Remain: 3days 23:19:16)

JOB_ID	JOB_NAME	STATUS	PROJECT	RSCGROUP	START_DATE	ELAPSE	TOKEN	NODE
290914	test	END	gt69	lecture9	04/18 12:46:24	00:01:44	-	1
290913	test	END	gt69	lecture9	04/18 12:46:06	00:02:07	-	1
290915	test	END	gt69	lecture9	04/18 12:49:26	00:00:59	-	1

(...)

```
[t69XYZ@obcx04 ~]$ pjstat --limit
```

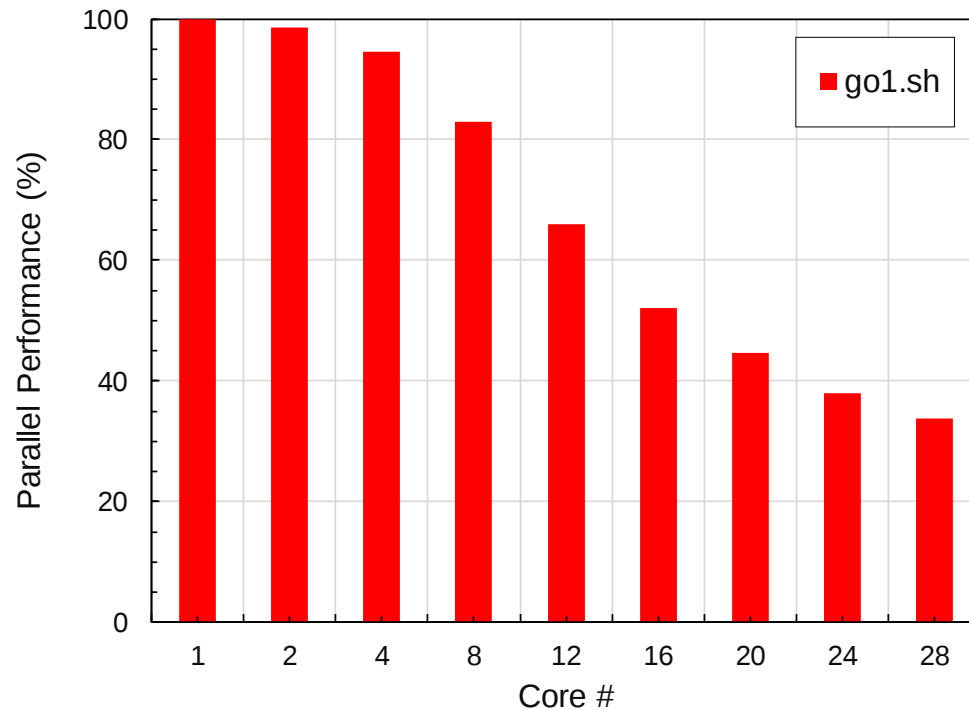
PROJECT	ACCEPT	RUN	BULK_RUN	NODE
gt69	0/ 80	0/ 20	0/ 256	0/ -

Results: Parallel Performance

NX=NY=NZ=128, go1.sh

Measurement: 5 times, best case

Parallel Performance = Speed-Up/Thread#



Thread #	sec	Speed-up
1	27.201	1.000
2	13.796	1.972
4	7.185	3.786
8	4.099	6.637
12	3.435	7.918
16	3.260	8.343
20	3.048	8.925
24	2.982	9.123
28	2.877	9.456



go1.sh

Only cores on a single socket used

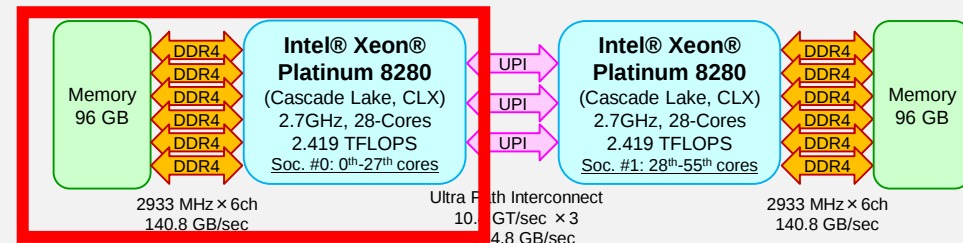
```
#!/bin/sh
#PJM -N "test1"
#PJM -L rscgrp=lecture9
#PJM -L node=1
#PJM --omp thread=24
#PJM -L elapse=00:15:00
#PJM -g gt69
#PJM -j
#PJM -e err
#PJM -o test1.lst
```

1, 2, 4, 8, 12, 16, 20, 24, 28

export KMP_AFFINITY=granularity=fine,compact

```
./sol20
./sol20
./sol20
./sol20
./sol20
```

Socket#0
Core #0-#27



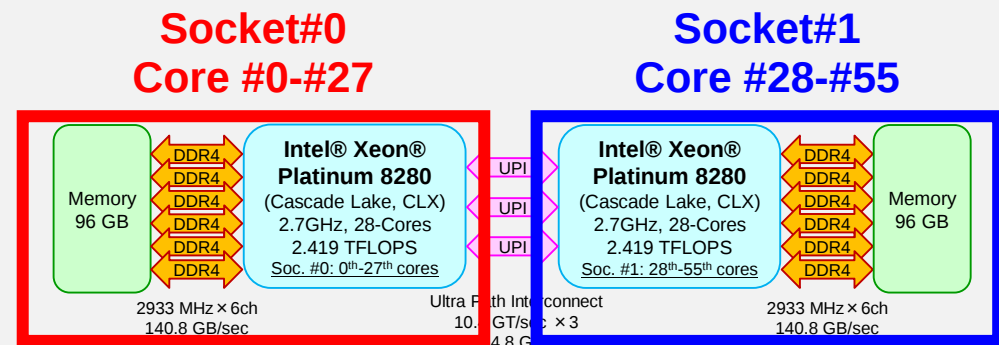
go2.sh

cores are randomly selected from 2 sockets

```
#!/bin/sh
#PJM -N "test2"
#PJM -L rscgrp=lecture9
#PJM -L node=1
#PJM --omp thread=24
#PJM -L elapse=00:15:00
#PJM -g gt69
#PJM -j
#PJM -e err
#PJM -o test1.1st
```

1, 2, 4, 8, 12, 16, 20, 24, 28

```
./sol20
./sol20
./sol20
./sol20
./sol20
```

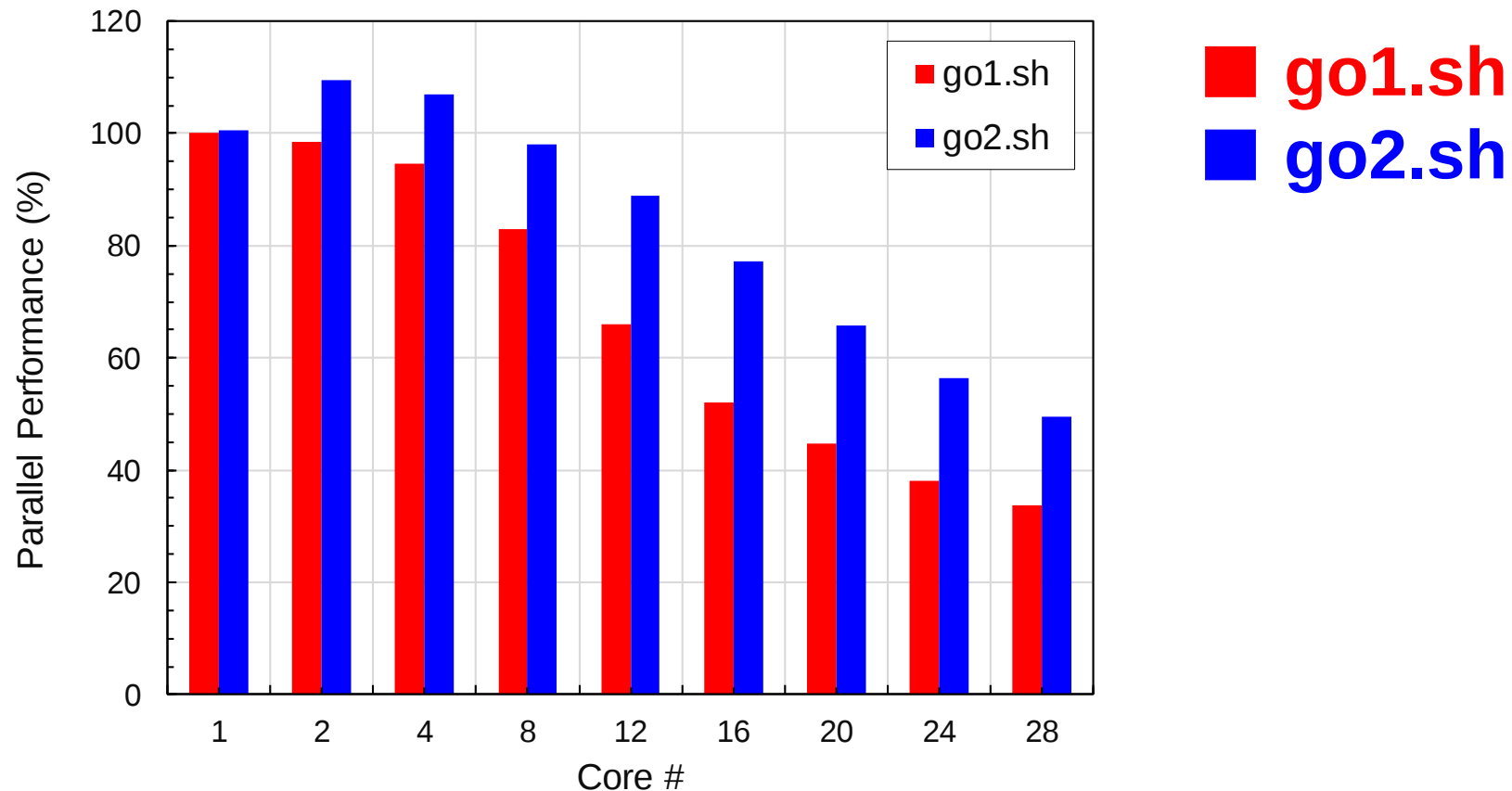


Results: Parallel Performance

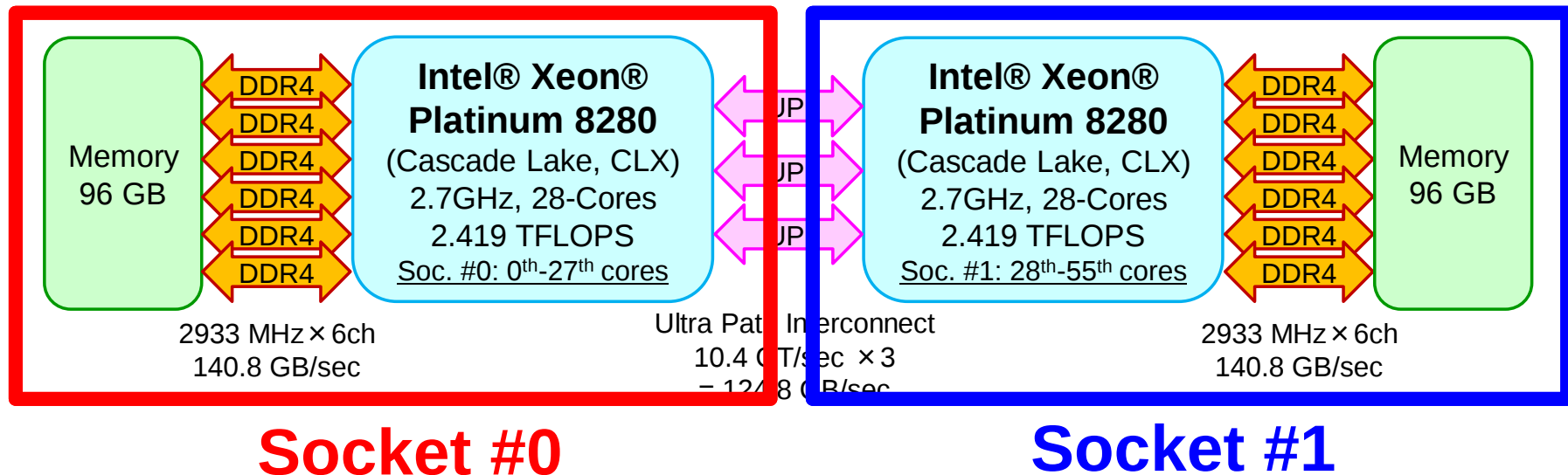
$NX=NY=NZ=128$

Measurement: 5 times, best case

“go2.sh” is generally better
based on “go1.sh” with 1 thread



OBCX



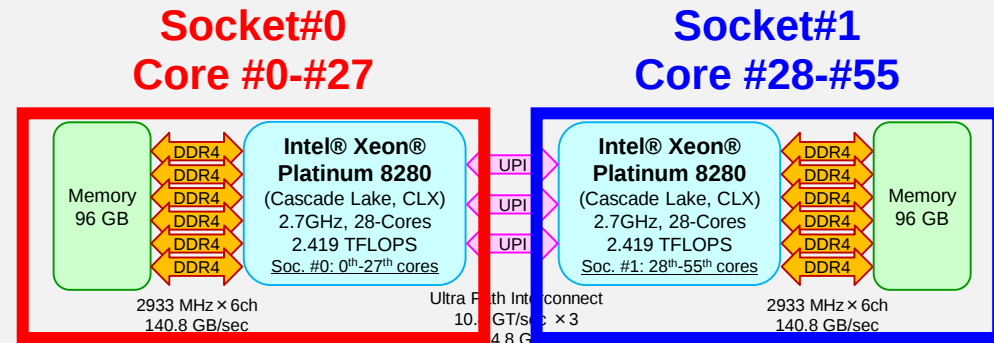
- Each Node of OBCX
 - 2 Sockets (CPU's) of Intel Broadwell-EP
 - Each socket has 28 cores
- Each core of a socket can access to the memory on the other socket : NUMA (Non-Uniform Memory Access)
- Utilization of the local memory is more efficient
- go2.sh
 - Number of used cores/socket could be smaller -> more efficient
 - But access to remote memory may happen -> NUMA, to be taught later

go3.sh

```
#!/bin/sh
#PJM -N "test2"
#PJM -L rscgrp=lecture9
#PJM -L node=1
#PJM --omp thread=24
#PJM -L elapse=00:15:00
#PJM -g gt69
#PJM -j
#PJM -e err
#PJM -o test3.1st
```

1, 2, 4, 8, 12, 16, 20, 24, 28

```
./sol20
```



```
export KMP_AFFINITY=granularity=fine,compact
```

```
./sol20
```

```
export KMP_AFFINITY=granularity=fine,balanced
```

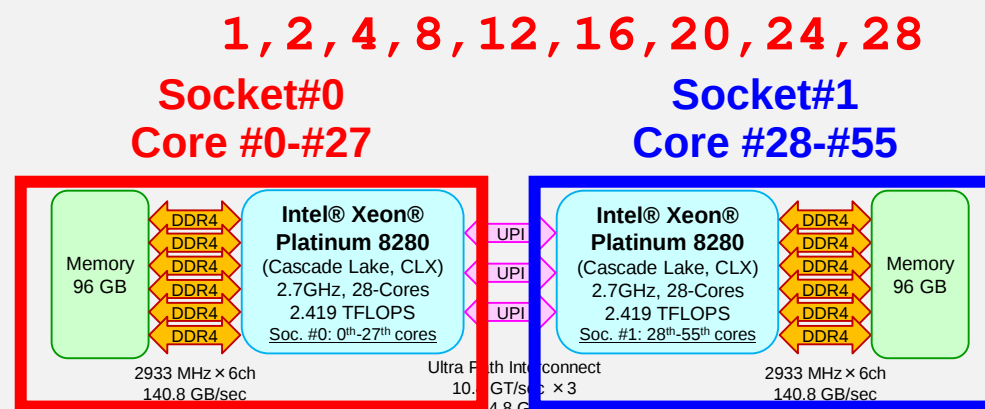
```
./sol20
```

```
export KMP_AFFINITY=granularity=fine,scatter
```

```
./sol20
```

go4.sh: “verbose” provides topology mapping (core-thread)

```
#!/bin/sh
#PJM -N "test2"
#PJM -L rscgrp=lecture9
#PJM -L node=1
#PJM --omp thread=24
#PJM -L elapse=00:15:00
#PJM -g gt69
#PJM -j
#PJM -e err
#PJM -o test3.lst
```



```
export KMP_AFFINITY=verbose
```

```
./sol20
```

```
export KMP_AFFINITY=granularity=fine,compact,verbose
```

```
./sol20
```

```
export KMP_AFFINITY=granularity=fine,balanced,verbose
```

```
./sol20
```

```
export KMP_AFFINITY=granularity=fine,scatter,verbose
```

```
./sol20
```

Results(none): 128^3 , 24 threads

2.044 sec.

```

OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225751 thread 0 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225753 thread 2 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225752 thread 1 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225754 thread 3 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225755 thread 4 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225756 thread 5 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225757 thread 6 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225758 thread 7 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225759 thread 8 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225760 thread 9 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225761 thread 10 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225762 thread 11 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225763 thread 12 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225764 thread 13 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225765 thread 14 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225766 thread 15 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225767 thread 16 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225768 thread 17 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225769 thread 18 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225770 thread 19 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225771 thread 20 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225772 thread 21 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225774 thread 23 bound to OS proc set 0-55
OMP: Info #251: KMP_AFFINITY: pid 225751 tid 225773 thread 22 bound to OS proc set 0-55

```

Socket#0
Core #0-#27

Socket#1
Core #28-#55

Thread ID

Core ID

Results(compact): 128³, 24 threads

2.978 sec., All threads on Soc.#0

```

OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225775 thread 0 bound to OS proc set 0
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225776 thread 1 bound to OS proc set 1
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225777 thread 2 bound to OS proc set 2
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225778 thread 3 bound to OS proc set 3
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225780 thread 5 bound to OS proc set 5
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225779 thread 4 bound to OS proc set 4
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225781 thread 6 bound to OS proc set 6
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225782 thread 7 bound to OS proc set 7
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225783 thread 8 bound to OS proc set 8
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225784 thread 9 bound to OS proc set 9
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225785 thread 10 bound to OS proc set 10
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225786 thread 11 bound to OS proc set 11
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225787 thread 12 bound to OS proc set 12
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225788 thread 13 bound to OS proc set 13
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225789 thread 14 bound to OS proc set 14
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225790 thread 15 bound to OS proc set 15
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225791 thread 16 bound to OS proc set 16
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225792 thread 17 bound to OS proc set 17
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225793 thread 18 bound to OS proc set 18
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225795 thread 20 bound to OS proc set 20
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225794 thread 19 bound to OS proc set 19
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225796 thread 21 bound to OS proc set 21
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225797 thread 22 bound to OS proc set 22
OMP: Info #251: KMP_AFFINITY: pid 225775 tid 225798 thread 23 bound to OS proc set 23

```

Socket#0
Core #0-#27

Socket#1
Core #28-#55

Thread ID

Core ID

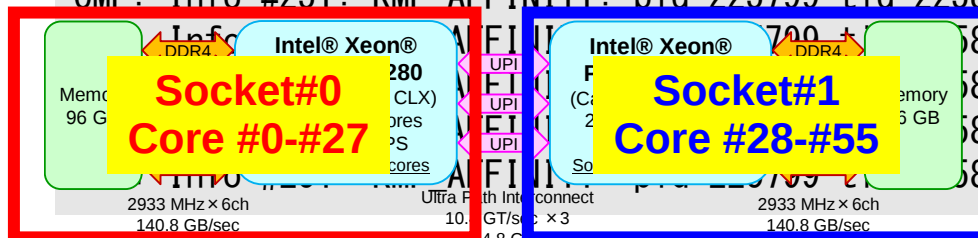
Results(balanced): 128^3 , 24 threads

2.007 sec., 0-11:Soc.#0, 12-23:Soc.#1

```

OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225799 thread 0 bound to OS proc set 0
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225800 thread 1 bound to OS proc set 1
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225801 thread 2 bound to OS proc set 2
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225802 thread 3 bound to OS proc set 3
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225803 thread 4 bound to OS proc set 4
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225804 thread 5 bound to OS proc set 5
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225805 thread 6 bound to OS proc set 6
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225806 thread 7 bound to OS proc set 7
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225807 thread 8 bound to OS proc set 8
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225808 thread 9 bound to OS proc set 9
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225809 thread 10 bound to OS proc set 10
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225810 thread 11 bound to OS proc set 11
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225811 thread 12 bound to OS proc set 28
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225812 thread 13 bound to OS proc set 29
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225813 thread 14 bound to OS proc set 30
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225814 thread 15 bound to OS proc set 31
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225815 thread 16 bound to OS proc set 32
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225816 thread 17 bound to OS proc set 33
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225817 thread 18 bound to OS proc set 34
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225818 thread 19 bound to OS proc set 35
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225819 thread 20 bound to OS proc set 36
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225820 thread 21 bound to OS proc set 37
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225821 thread 22 bound to OS proc set 38
OMP: Info #251: KMP_AFFINITY: pid 225799 tid 225822 thread 23 bound to OS proc set 39

```



Thread ID

Core ID

Results(scatter): 128^3 , 24 threads

2.175 sec.

```

OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225823 thread 0 bound to OS proc set 0
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225824 thread 1 bound to OS proc set 28
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225825 thread 2 bound to OS proc set 1
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225826 thread 3 bound to OS proc set 29
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225827 thread 4 bound to OS proc set 2
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225828 thread 5 bound to OS proc set 30
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225829 thread 6 bound to OS proc set 3
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225830 thread 7 bound to OS proc set 31
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225831 thread 8 bound to OS proc set 4
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225832 thread 9 bound to OS proc set 32
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225833 thread 10 bound to OS proc set 5
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225834 thread 11 bound to OS proc set 33
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225835 thread 12 bound to OS proc set 6
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225836 thread 13 bound to OS proc set 34
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225837 thread 14 bound to OS proc set 7
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225838 thread 15 bound to OS proc set 35
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225839 thread 16 bound to OS proc set 8
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225840 thread 17 bound to OS proc set 36
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225841 thread 18 bound to OS proc set 9
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225842 thread 19 bound to OS proc set 37
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225843 thread 20 bound to OS proc set 10
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225844 thread 21 bound to OS proc set 38
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225845 thread 22 bound to OS proc set 11
OMP: Info #251: KMP_AFFINITY: pid 225823 tid 225846 thread 23 bound to OS proc set 39

```

Socket#0
Core #0-#27

Socket#1
Core #28-#55

Thread ID

Core ID

Exercises

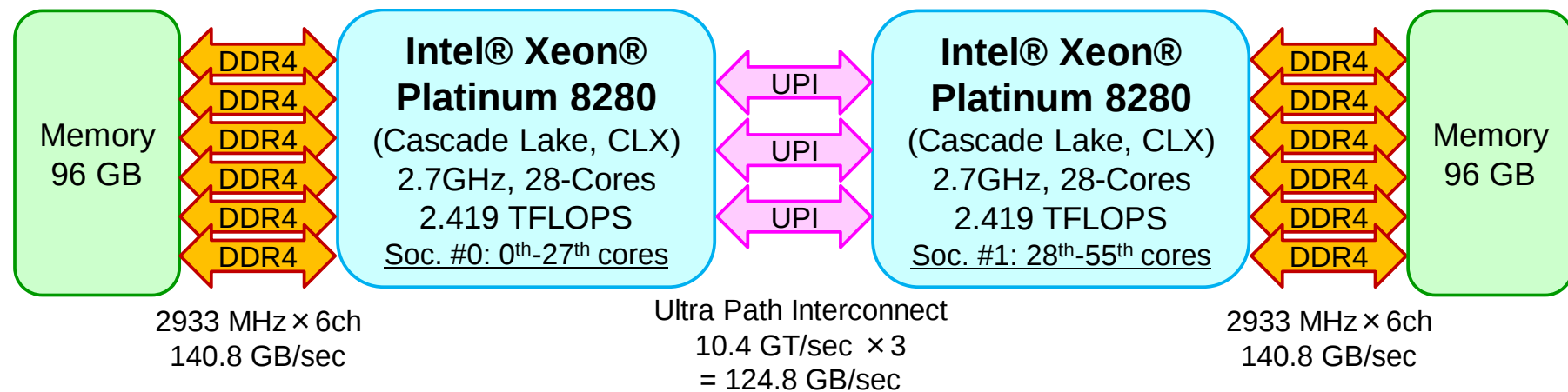
- Effect of problem size (NX, NY, NZ)
- Effect of Thread # (OMP_NUM_THREADS: 1-28)

- OpenMP
- Login to OBCX
- Parallel Version of the Code by OpenMP
- **STREAM**
- Data Dependency

Why less than 28x ?

- Memory Saturation (メモリ飽和)
- 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 large
- Why go2.sh is better ?

Category	Capacity	X-Way Set Associative	Cache Line
L1\$Data	32 KB/core	8-Way	64B
L1\$Instruction	32 KB/core	8-Way	64B
L2	1.00 MB/core	16-Way	64B
L3	38.5 MB/socket	11-Way	64B



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; j++) {
    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 (in-direct access), 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: $c(i) = a(i) + s * b(i)$

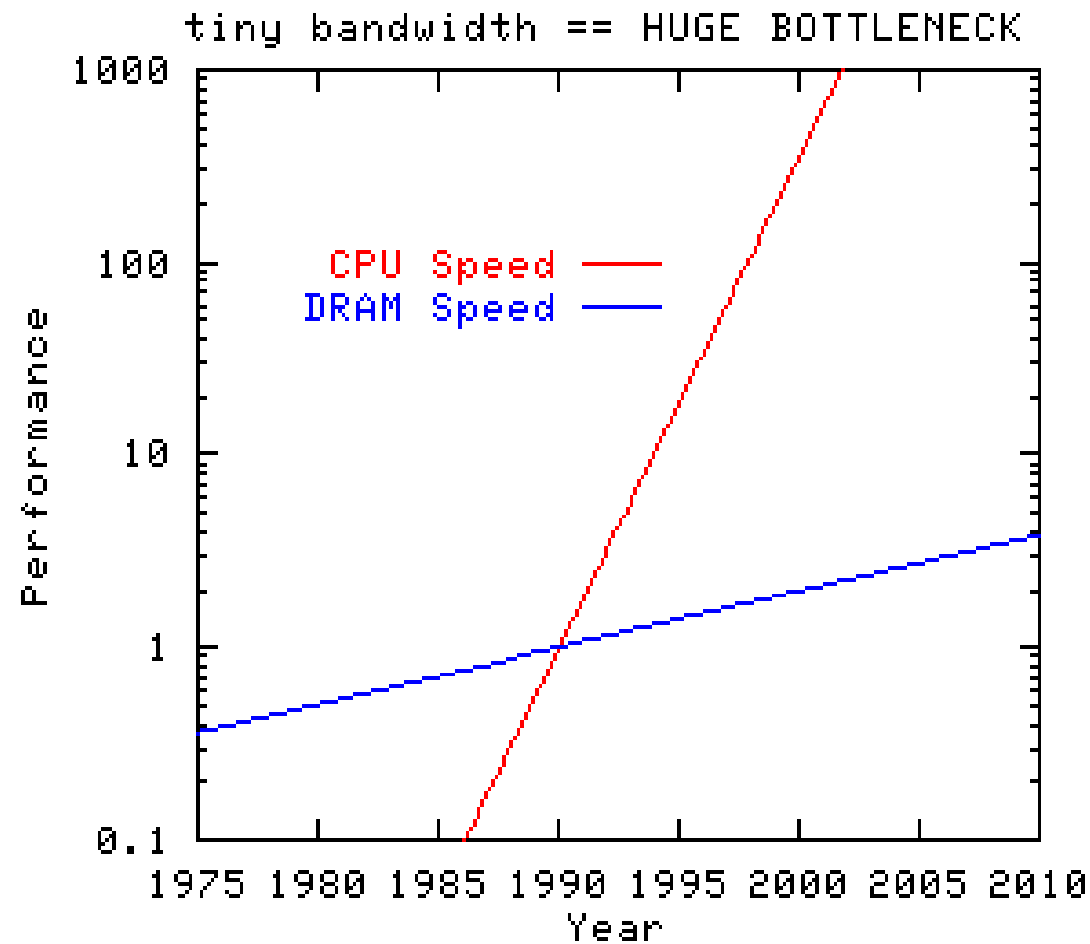
Double precision appears to have 16 digits of accuracy
Assuming 8 bytes per DOUBLE PRECISION word

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



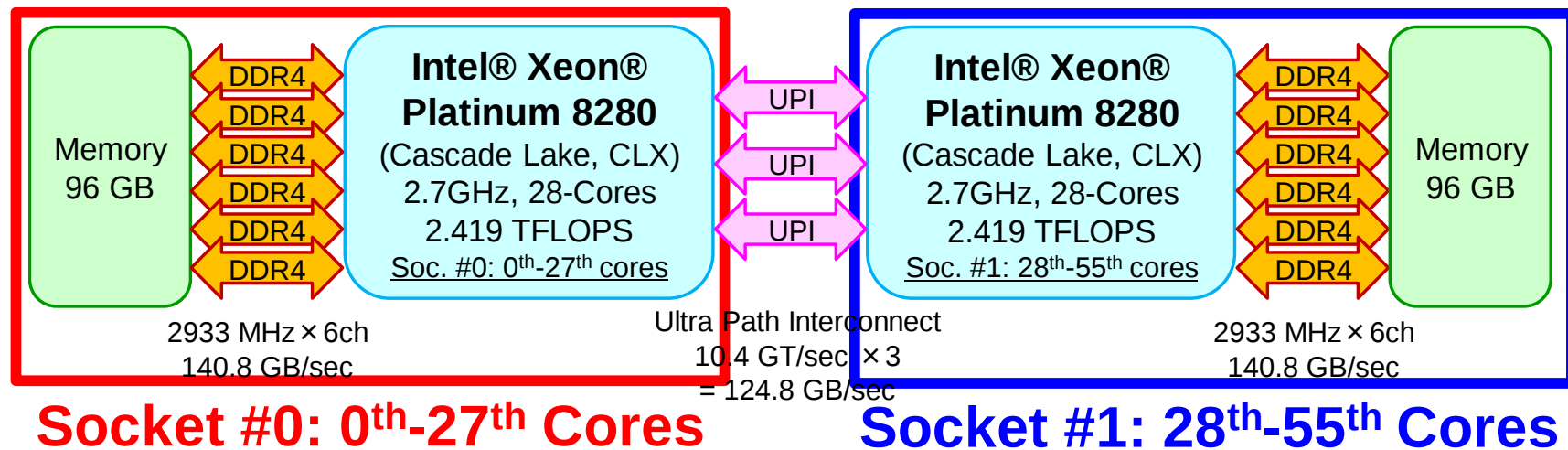
Copy, Compile and Run MPI Version

```
>$ cd /work/gt69/t69XXX/multicore/stream
```

```
>$ mpiifort -align array64byte -O3 -axCORE-AVX512 stream.f -o stream
```

Please add the option “-no-multibyte-chars”, if you have any problems in compiling

```
>$ pjsub XXX.sh
```



s01.sh: Use 1 core

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture9
#PJM -L node=1
#PJM --mpi proc=1
#PJM -L elapse=00:15:00
#PJM -g gt69
#PJM -j
#PJM -e err
#PJM -o s01.lst
```

```
mpiexec.hydra -n ${PJM_MPI_PROC} ./stream
```

```
export I_MPI_PIN_PROCESSOR_LIST=0
```

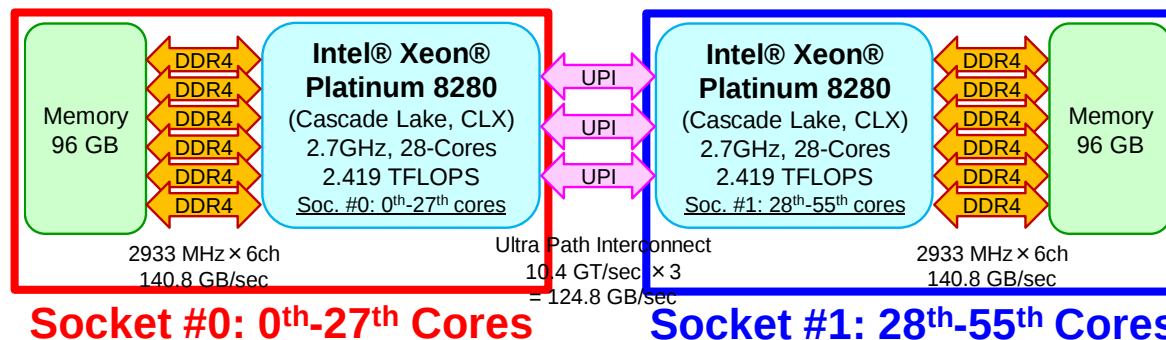
```
mpiexec.hydra -n ${PJM_MPI_PROC} ./stream
```

Cores are randomly
selected

■ go2.sh
random

Core are specified

■ go1.sh
compact



s16.sh: Use 16 cores

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture9
#PJM -L node=1
#PJM --mpi proc=16
#PJM -L elapse=00:15:00
#PJM -g gt69
#PJM -j
#PJM -e err
#PJM -o s16.lst
```

```
mpiexec.hydra -n ${PJM_MPI_PROC} ./stream
```

```
export I_MPI_PIN_PROCESSOR_LIST=0-15
```

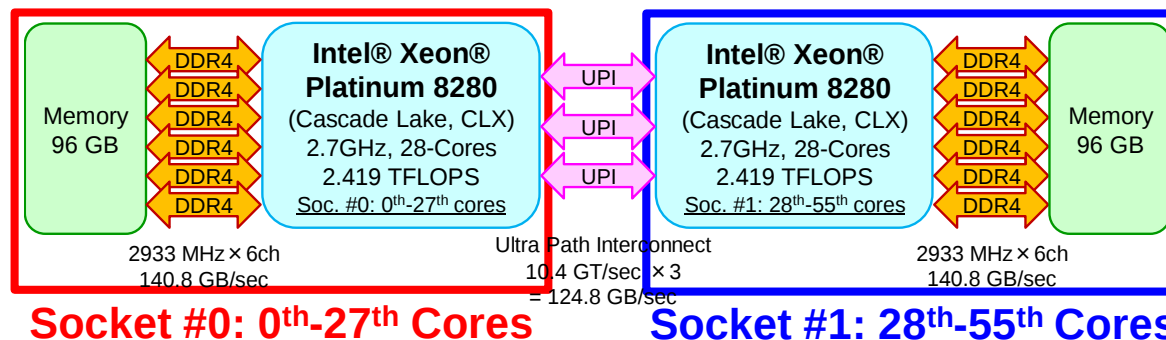
```
mpiexec.hydra -n ${PJM_MPI_PROC} ./stream
```

Cores are randomly
selected

■ go2.sh
random

Core are specified

■ go1.sh
compact



s32.sh: Use 32 cores

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture9
#PJM -L node=1
#PJM --mpi proc=32
#PJM -L elapse=00:15:00
#PJM -g gt69
#PJM -j
#PJM -e err
#PJM -o s32.lst
```

```
mpiexec.hydra -n ${PJM_MPI_PROC} ./stream
```

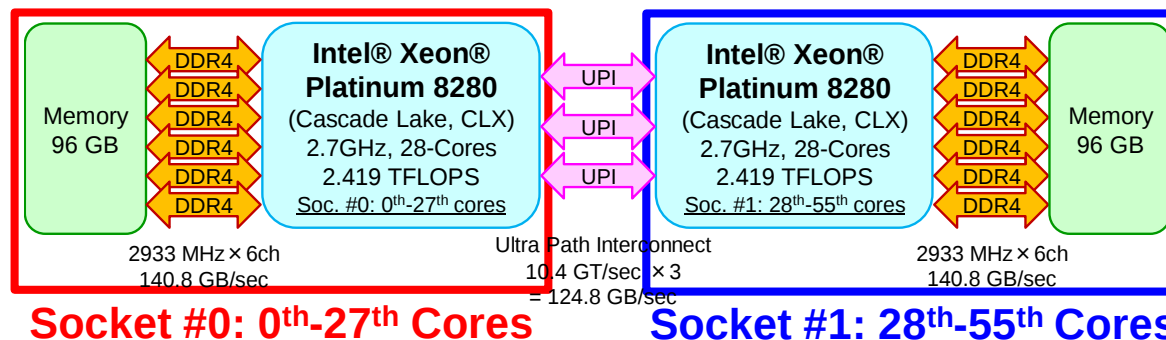
```
export I_MPI_PIN_PROCESSOR_LIST=0-15,28-43
mpiexec.hydra -n ${PJM_MPI_PROC} ./stream
```

Cores are randomly
selected

■ go2.sh
random

Core are specified

■ go1.sh
compact



s48.sh: Use 48 cores

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture9
#PJM -L node=1
#PJM --mpi proc=48
#PJM -L elapse=00:15:00
#PJM -g gt69
#PJM -j
#PJM -e err
#PJM -o s48.lst
```

```
mpiexec.hydra -n ${PJM_MPI_PROC} ./stream
```

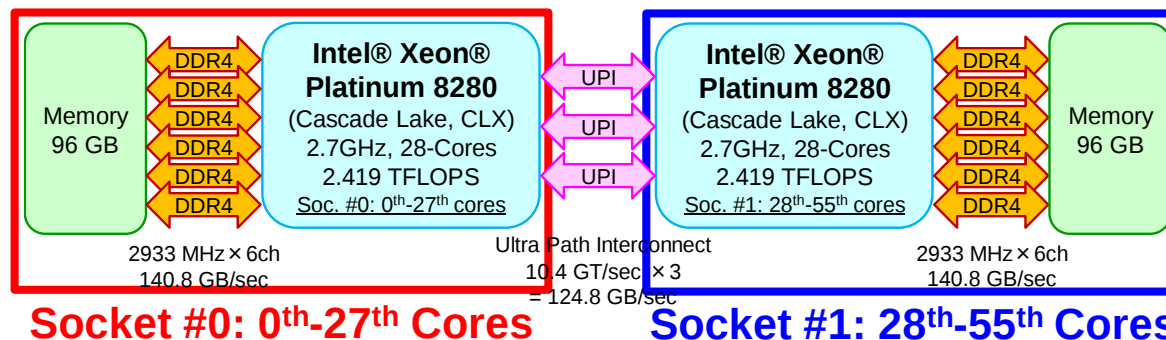
```
export I_MPI_PIN_PROCESSOR_LIST=0-23,28-51
mpiexec.hydra -n ${PJM_MPI_PROC} ./stream
```

Cores are randomly
selected

■ go2.sh
random

Core are specified

■ go1.sh
compact



s56.sh: Use 56 cores

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture9
#PJM -L node=1
#PJM --mpi proc=56
#PJM -L elapse=00:15:00
#PJM -g gt69
#PJM -j
#PJM -e err
#PJM -o s56.lst
```

```
mpiexec.hydra -n ${PJM_MPI_PROC} ./stream
```

```
export I_MPI_PIN_PROCESSOR_LIST=0-55
```

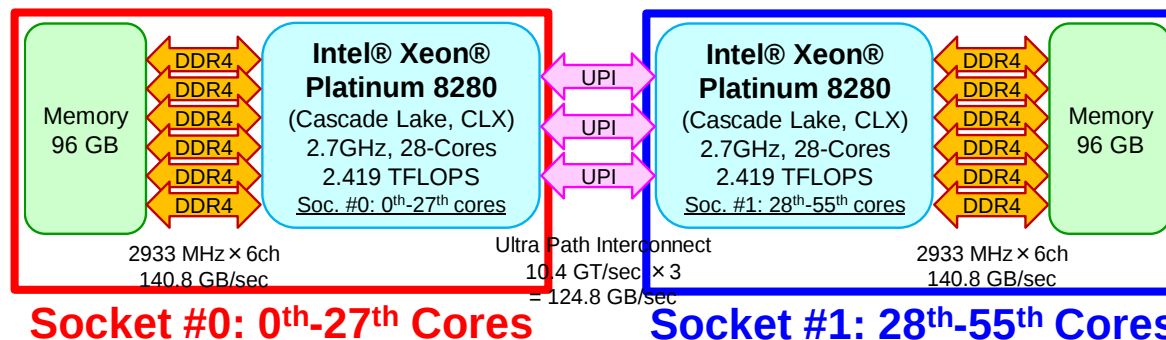
```
mpiexec.hydra -n ${PJM_MPI_PROC} ./stream
```

Cores are randomly
selected

■ go2.sh
random

Core are specified

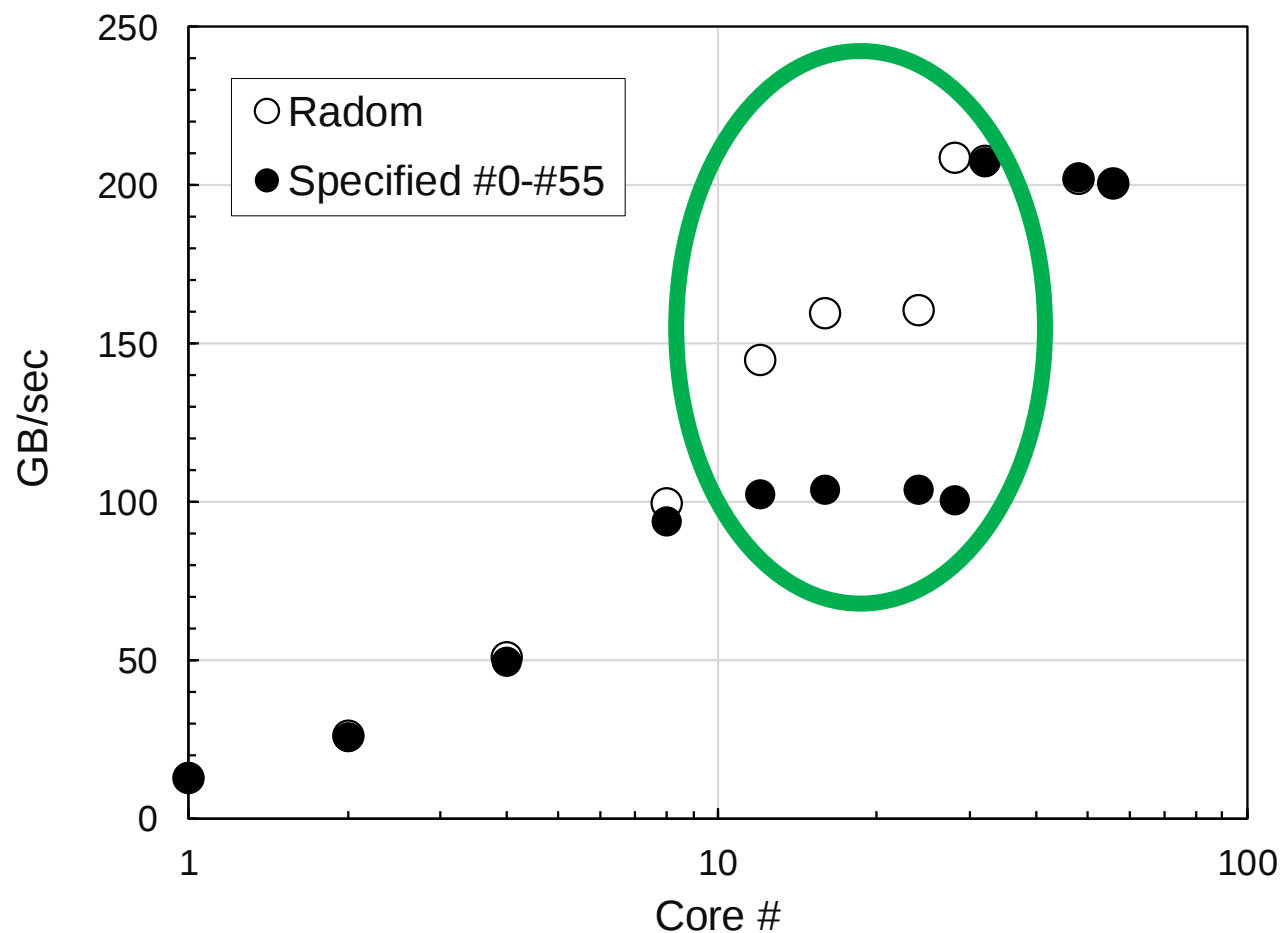
■ go1.sh
compact



Triad on a Single Node of OBCX

Peak is 281.57 GB/sec.

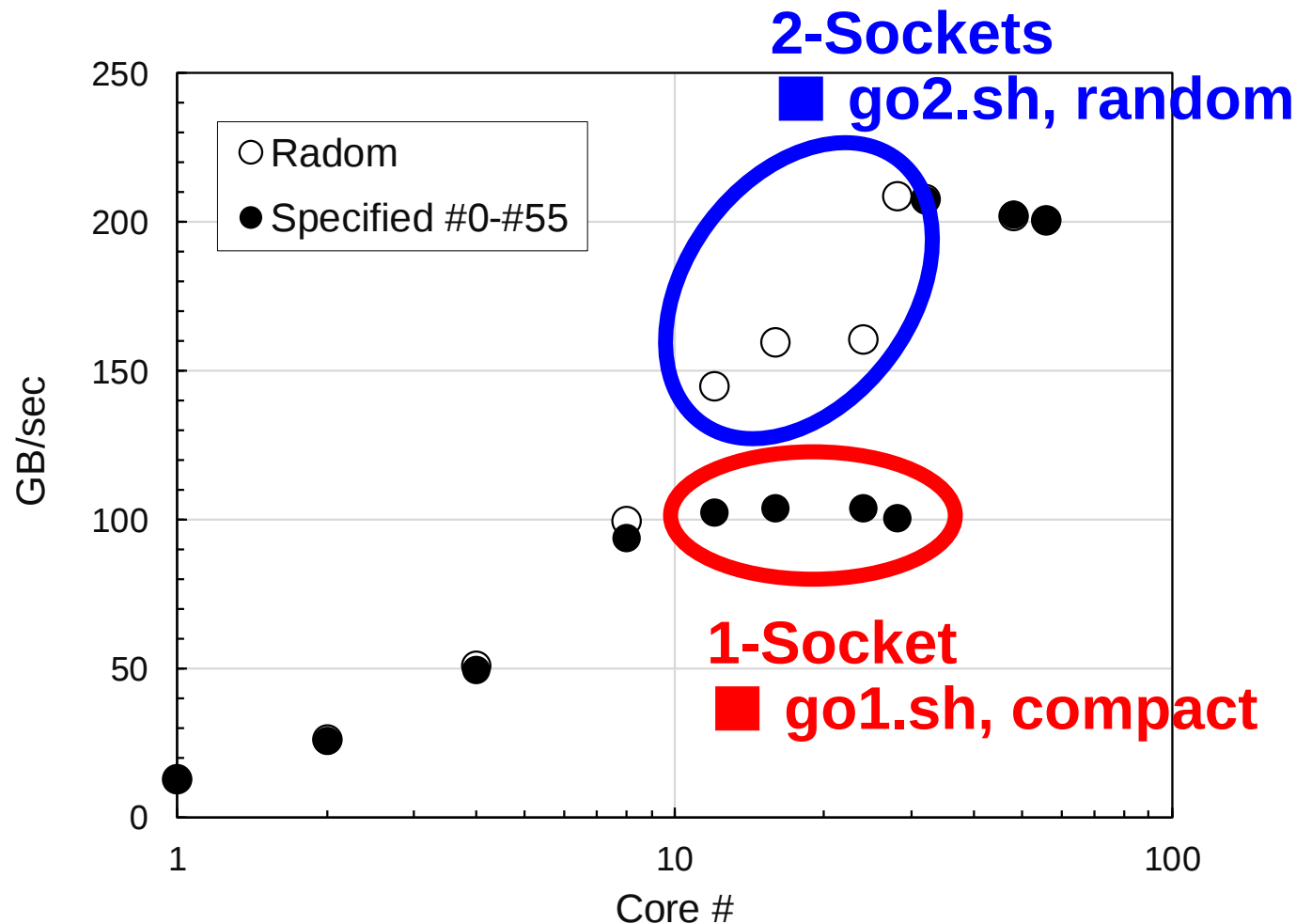
“Random” is much better at (12-28 cores)

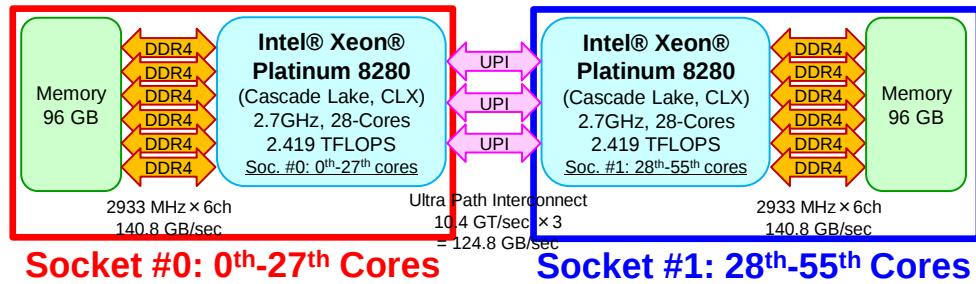


Triad on a Single Node of OBCX

Peak is 281.57 GB/sec.

“Random” is much better at (12-28 cores)





Socket #	Core #	Random go2.sh random	Specified go1.sh compact
1	1	1.000	1.000
	2	1.998	1.963
	4	3.912	3.809
	6	5.792	5.682
	8	7.615	7.177
	12	11.089	7.844
	16	12.214	7.968
	24	12.278	7.945
2	28	15.962	7.705
	32	15.901	15.862
	48	15.452	15.497
	56	15.370	15.358

Triad on a Single Node of OBCX

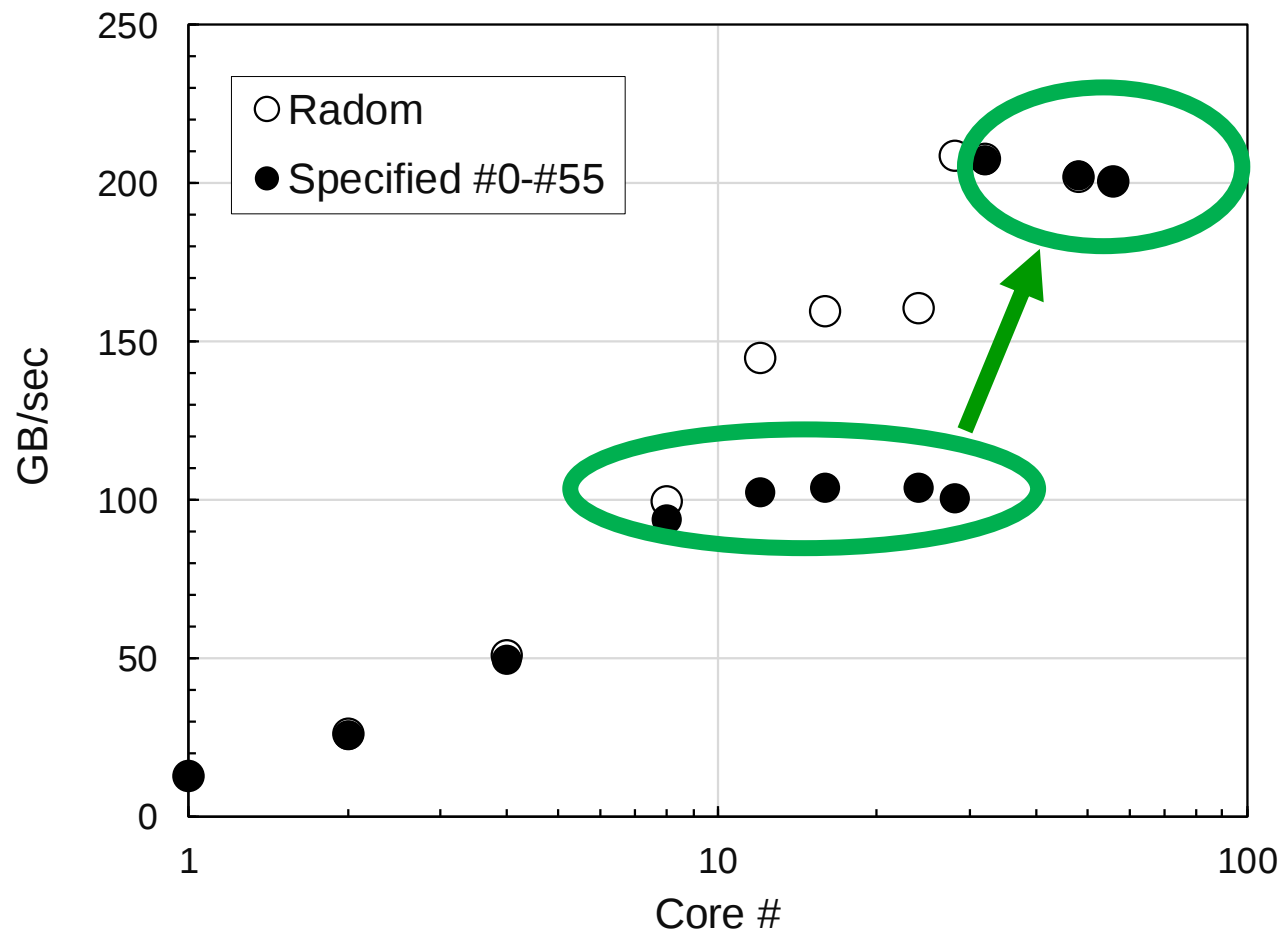
Speed-Up based on Performance with 1 core

- Number of Memory Channels/Socket = 6
 - 6 memory chips can be loaded on each socket
 - Linear speed-up up to 6 cores (processes) on each socket
- 12-28 cores
 - Random: 2-Sockets are used (6-14 core/socket)
 - Specified: 1-Socket

Triad on a Single Node of OBCX

Peak is 281.57 GB/sec.

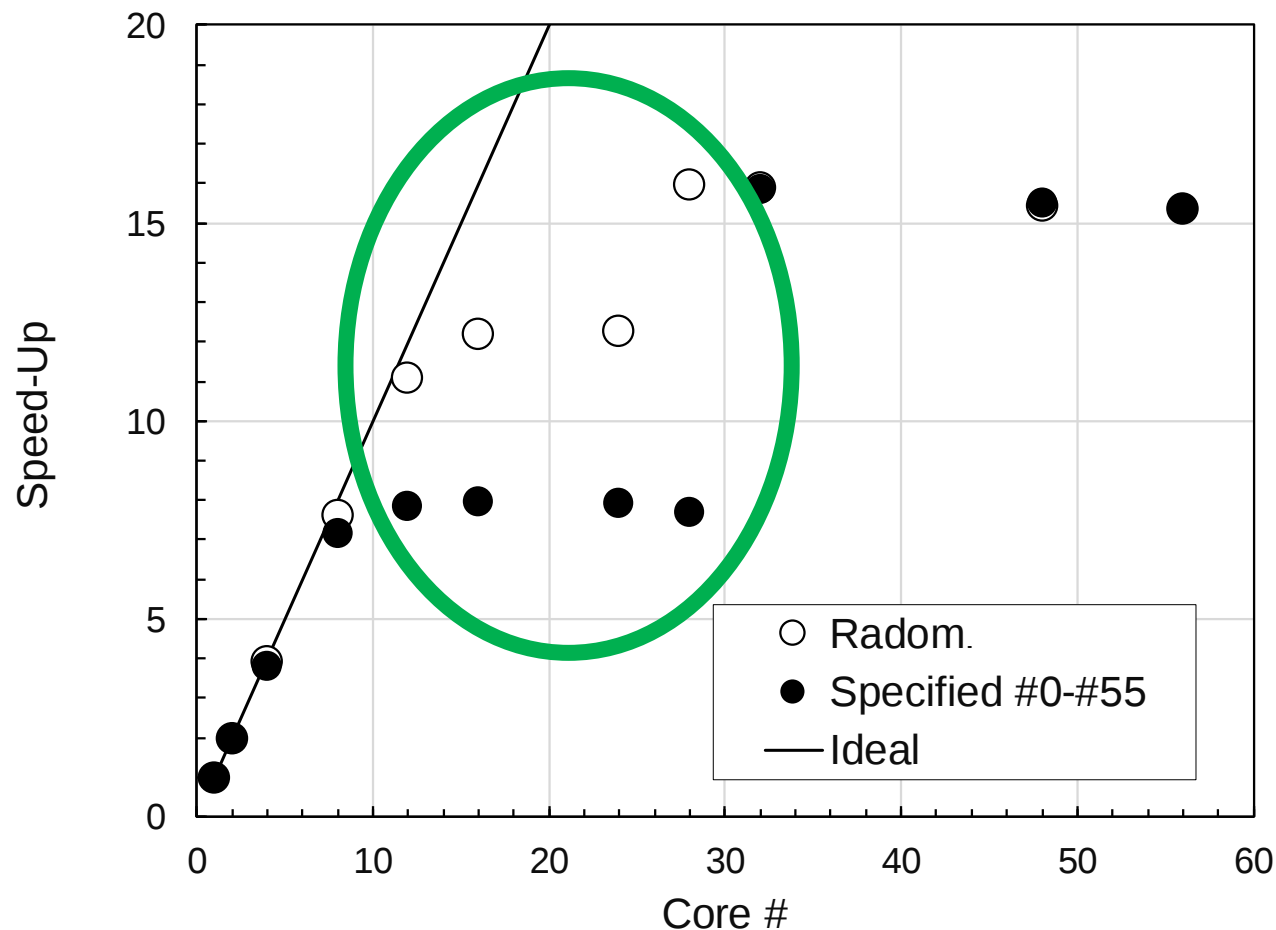
● : Memory BW is constant with 8-28 cores (saturated),
Doubled with 32-56 cores



Triad on a Single Node of OBCX

Peak is 281.57 GB/sec.

“Random” is much better at (12-28 cores)



Exercises

- Running the code
- Try various number of processes (1-56)
- OpenMP-version and Single PE version are available
 - Fortran, C
 - Web-site of STREAM
 - <http://www.cs.virginia.edu/stream/>

- OpenMP
- Login to OBCX
- Parallel Version of the Code by OpenMP
- STREAM
- **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];
}
```

Matrix-Vector Multiply: Easy to be Parallelized $\{q\}=[A]\{p\}$

$\{q\}$: LHS: Updated

$\{p\}$: RHS: No change

```
#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;
}
```

IC Factorization

Four Thread Parallel Operation

“icel” starts at 0

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

```
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;
}
```

“itemL” starts at 1

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

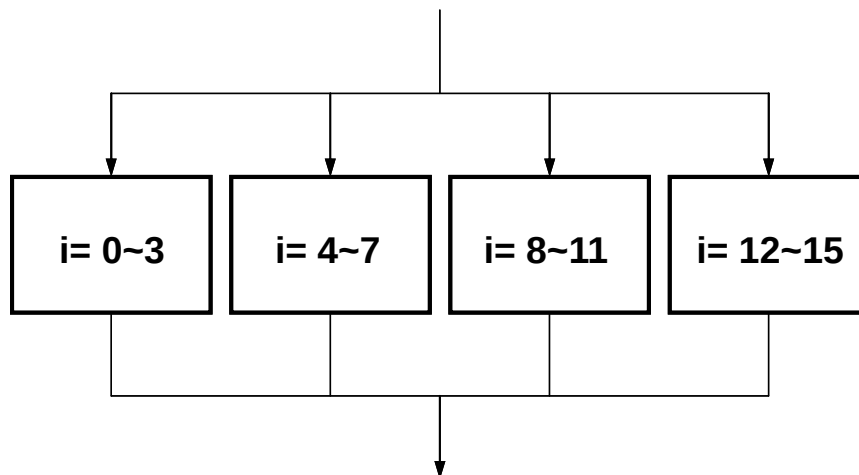
IC Factorization

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



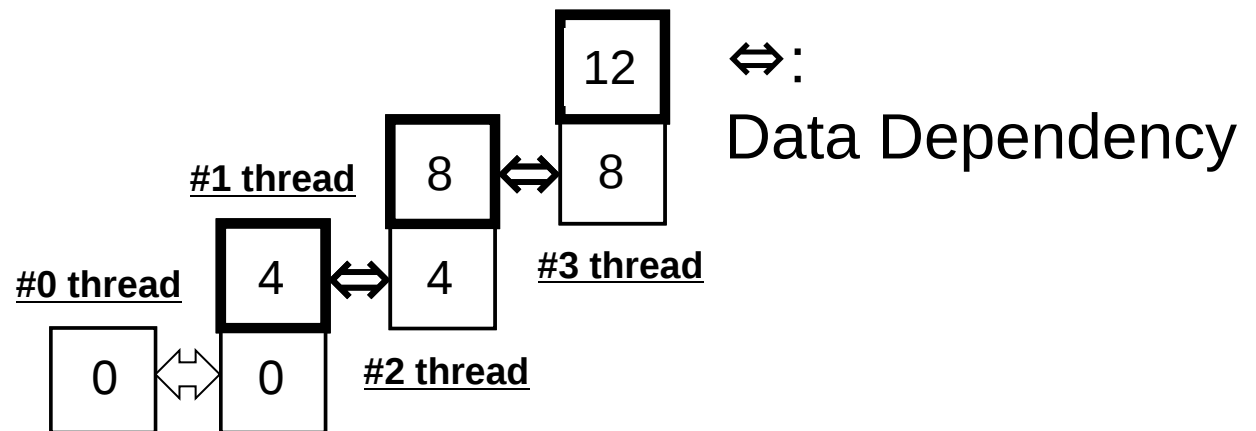
These four threads are executed simultaneously.

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