

Introduction to Parallel Programming for Multicore/Manycore Clusters

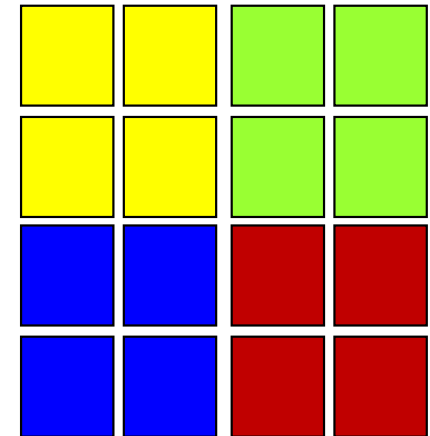
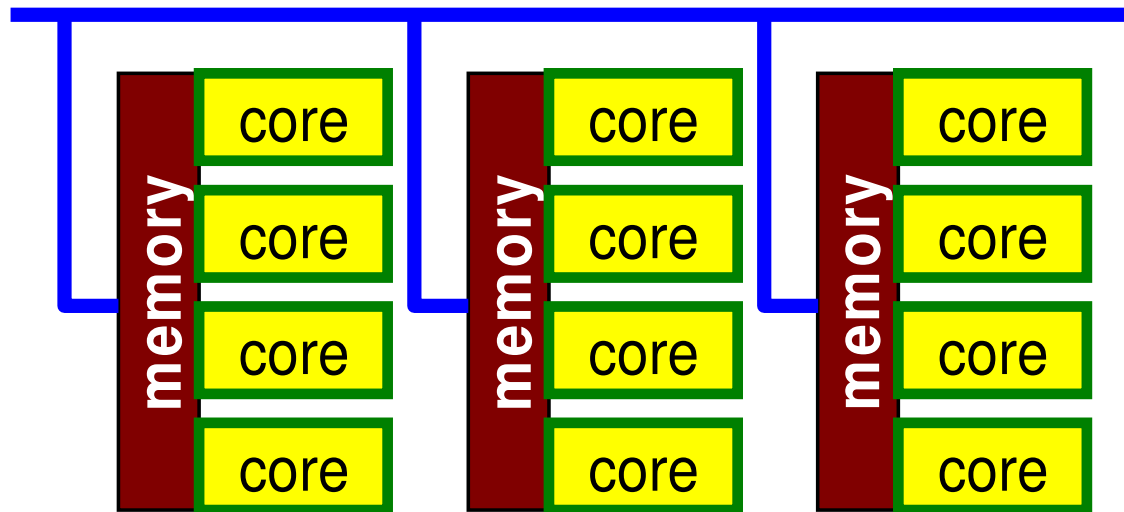
Part A2: Introduction to OpenMP

Kengo Nakajima
Information Technology Center
The University of Tokyo

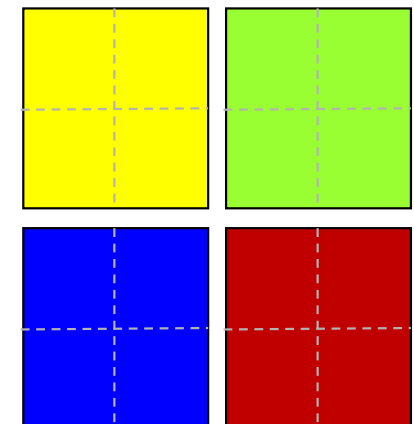
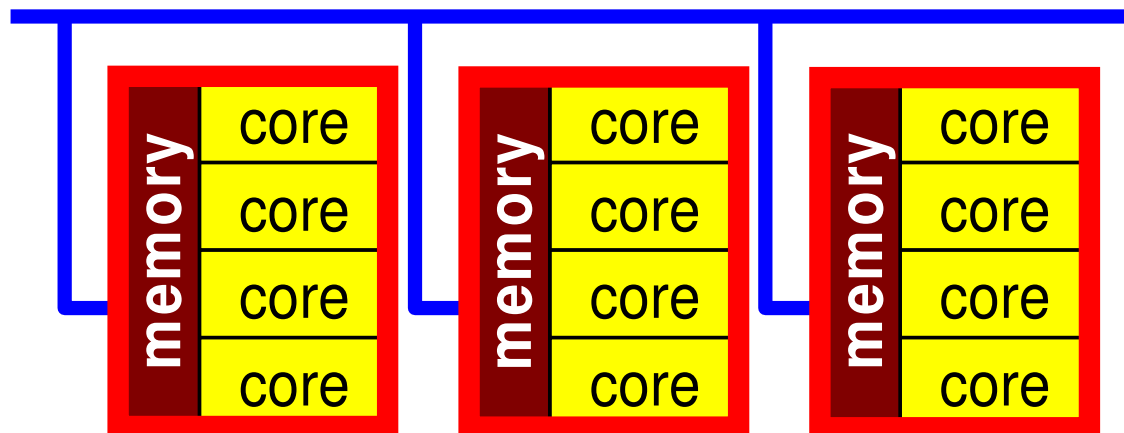
- OpenMP
- Login to Wisteria/BDEC-01
- Parallel Code by OpenMP (0): up to 12 cores
- Parallel Code by OpenMP (1): First Touch
- Parallel Code by OpenMP (2): +ELL
- Parallel Code by OpenMP (3): reduced omp-parallel
- Parallel Code by OpenMP (4): Further Optimization (Fortran only)

Flat MPI vs. Hybrid

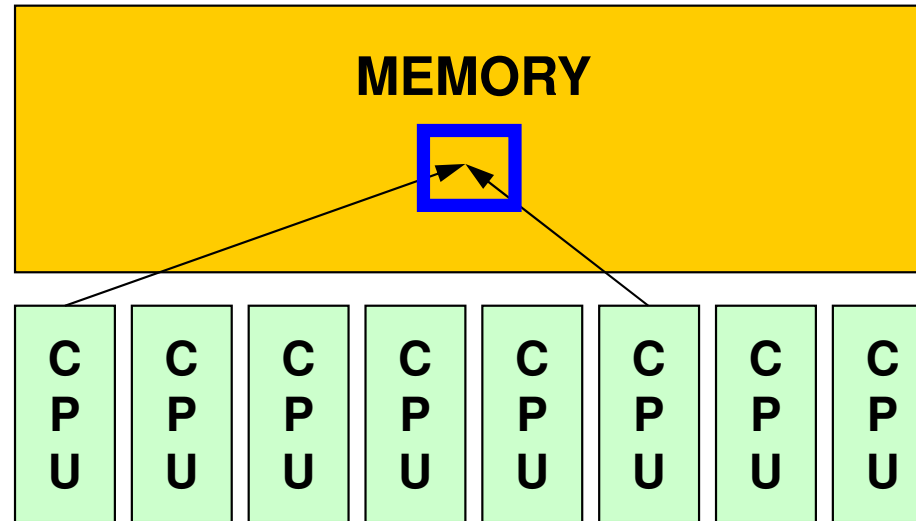
Flat-MPI: Each Core -> Independent



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>

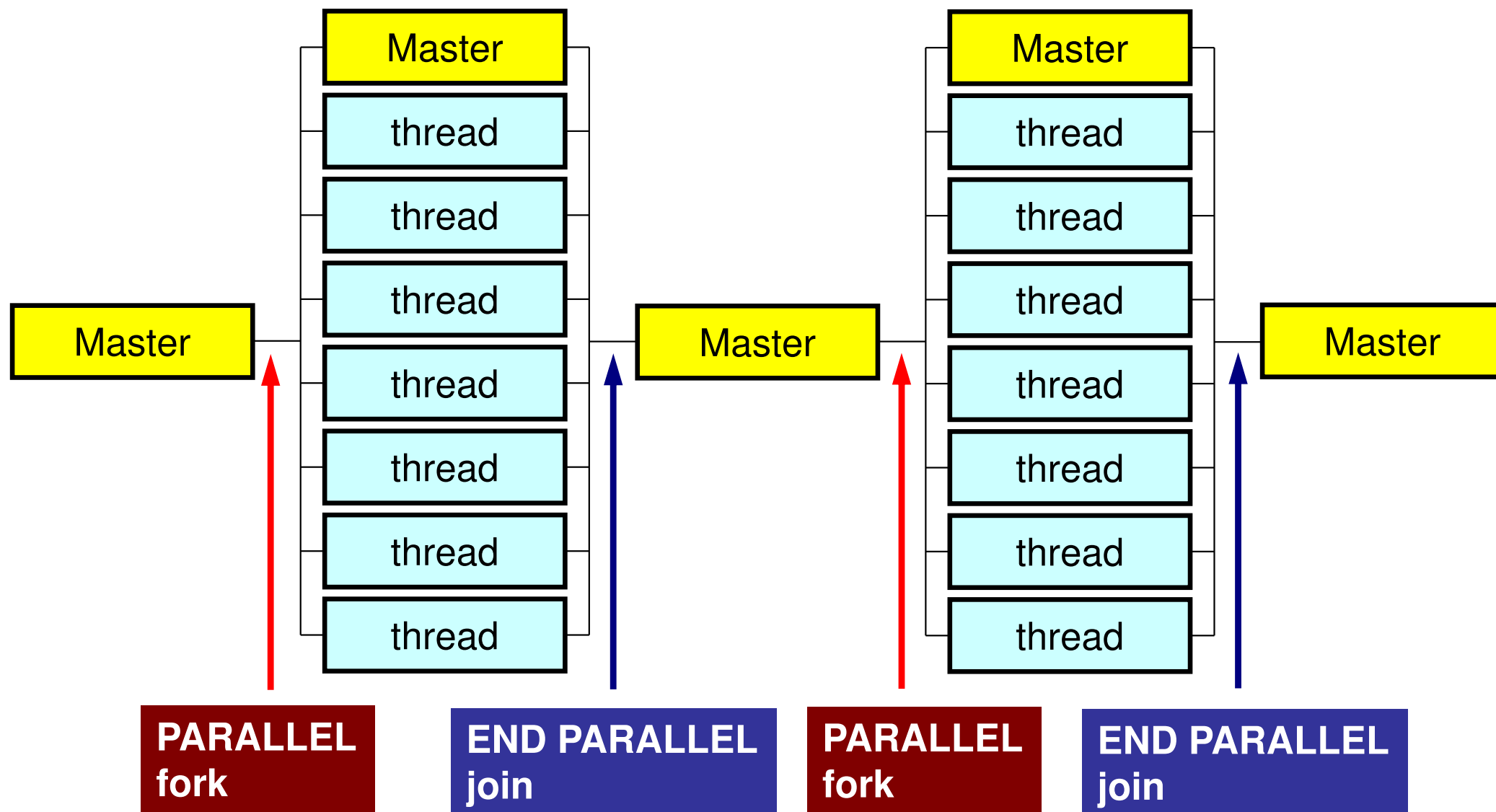
- An API (Application Programming Interface) for multi-platform shared-memory parallel programming in C/C++ and Fortran
 - Current version: 4.X (5.0 is already announced)
 - GPU, Accelerators: close to OpenACC
- Background
 - Merger of Cray and SGI in 1996
 - Separated later, ... but both are now merged 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 ?

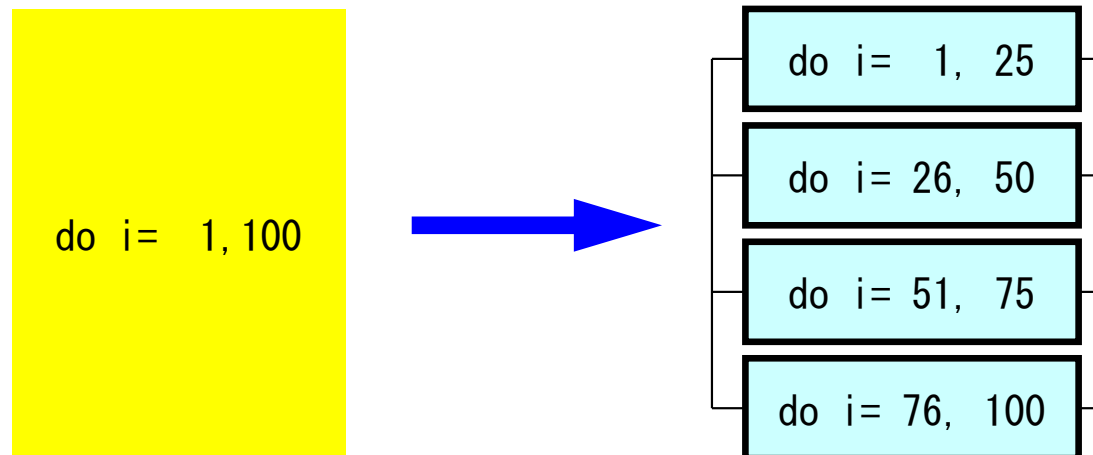
- `bash(.bashrc)`

- `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

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

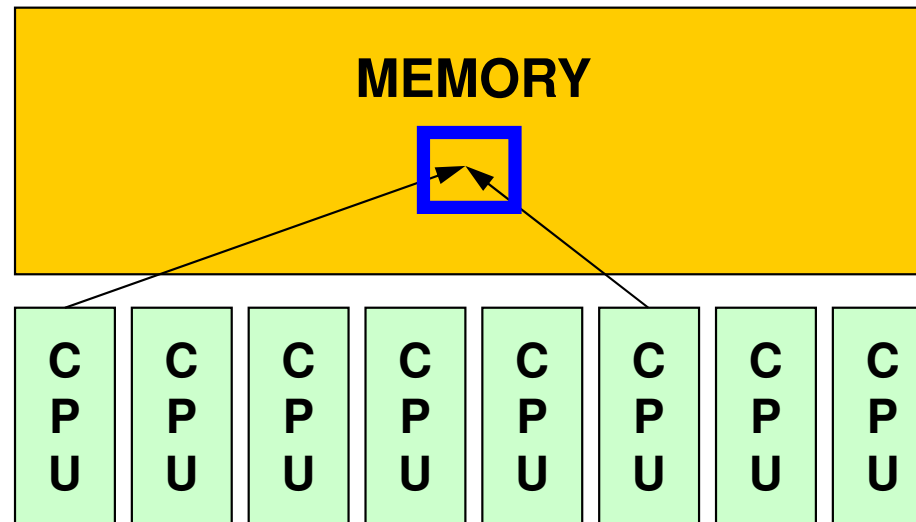
Matrix/Vector Products

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

Features of OpenMP

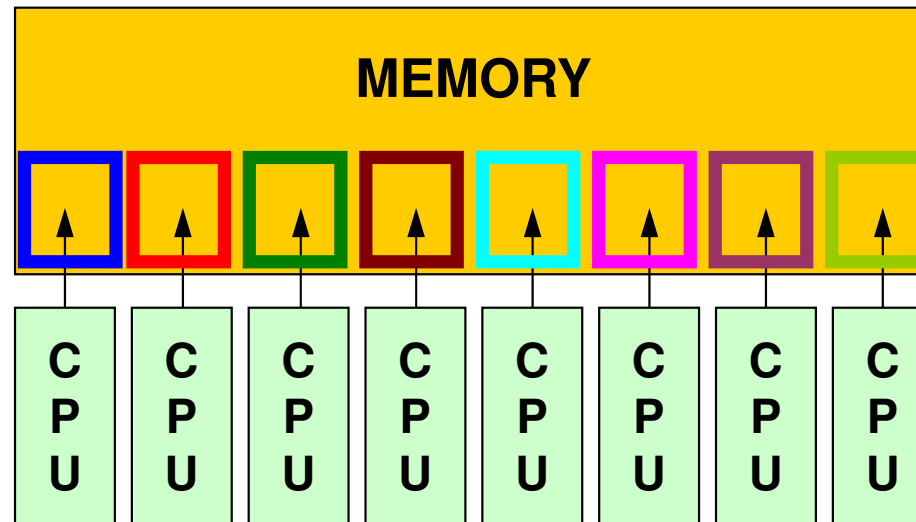
- Directives
 - Loops right after the directives are parallelized.
 - If the compiler does not support OpenMP, directives are considered as just comments.
- Nothing happen without explicit directives
 - Different from “automatic parallelization/vectorization”
 - Something wrong may happen by un-proper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

- “!omp parallel do”-“!omp end parallel do”
- 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 PCG/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];  
}
```

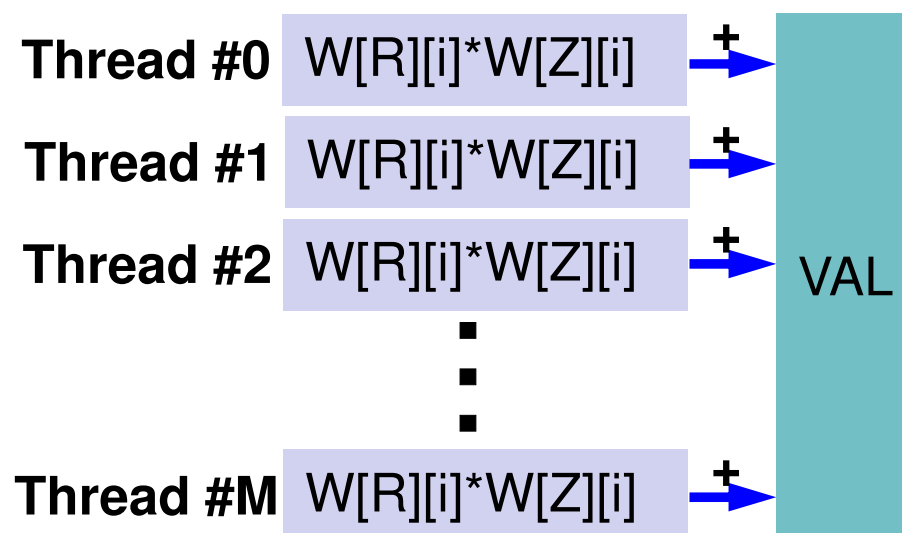
OpenMP for Dot Products

```
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.

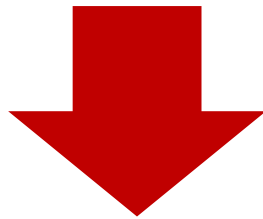
Efficiency is not necessarily good,
but users can specify thread for
each component of data.

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

```
INDEX[0]= 0  
INDEX[1]= 25  
INDEX[2]= 50  
INDEX[3]= 75  
INDEX[4]= 100
```

Matrix-Vector Multiply

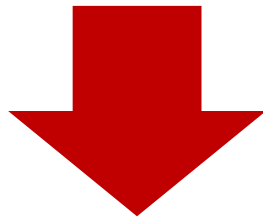
```
for (i=0; i<N; i++) {  
    VAL = D[i] * W[P][i];  
    for (j=indexLU[i]; j<indexLU[i+1]; j++) {  
        VAL += AMAT[j] * W[P][itemLU[j]];  
    }  
    W[Q][i] = VAL;  
}
```



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

Matrix-Vector Multiply

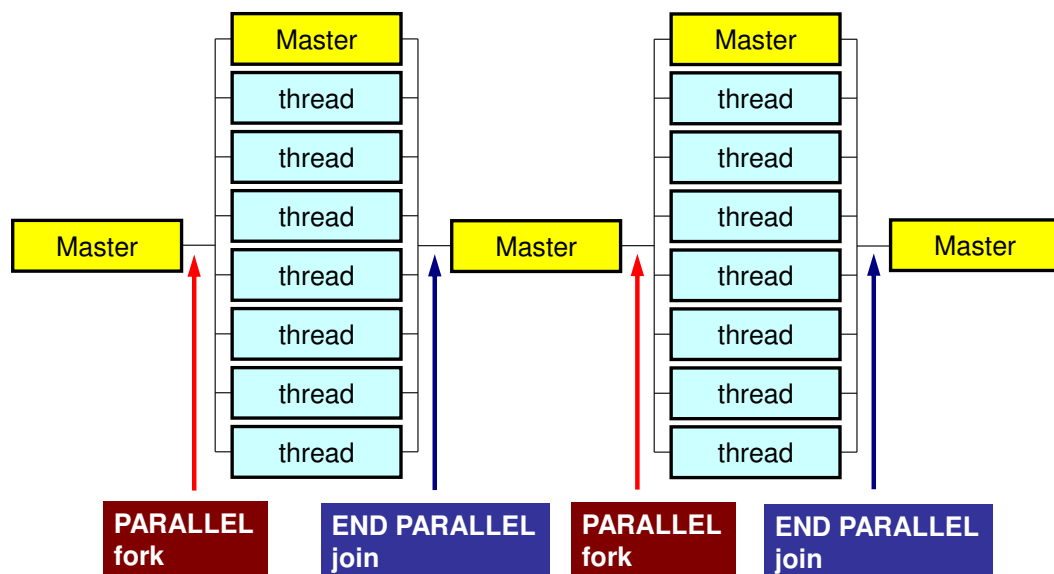
```
for (i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for (j=indexLU[i]; j<indexLU[i+1]; j++) {
        VAL += AMAT[j] * W[P][itemLU[j]];
    }
    W[Q][i] = VAL;
}
```



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


omp parallel (do)

- “omp parallel-omp end parallel” = “fork-join”
- If you have many loops, these “fork-join’s” cause overheads
- **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 Wisteria/BDEC-01](#)
- Parallel Code by OpenMP (0): up to 12 cores
- Parallel Code by OpenMP (1): First Touch
- Parallel Code by OpenMP (2): +ELL
- Parallel Code by OpenMP (3): reduced omp-parallel
- Parallel Code by OpenMP (4): Further Optimization (Fortran only)

- OpenMP
- Login to Wisteria/BDEC-01
- **Parallel Code by OpenMP (0): up to 12 cores**
- Parallel Code by OpenMP (1): First Touch
- Parallel Code by OpenMP (2): +ELL
- Parallel Code by OpenMP (3): reduced omp-parallel
- Parallel Code by OpenMP (4): Further Optimization (Fortran only)

Target for Parallelization

- “Parallelization” of the FVM code
- Just inserting OpenMP directives
 - “poi_gen.f/c”(poi_gen),
 - “solver_PCG.f/c” (solve_PCG)

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

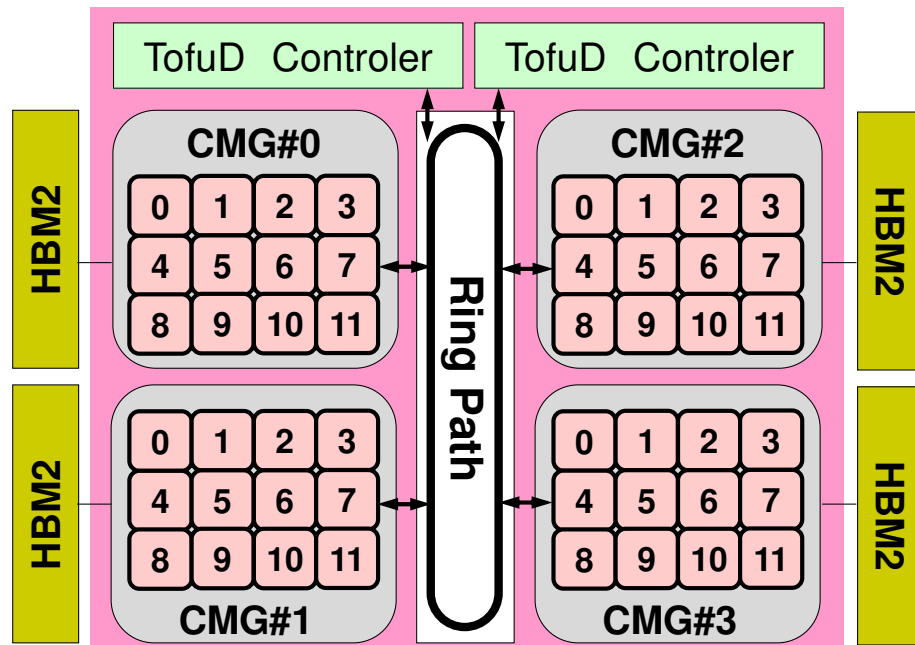
$$[M]^{-1} = [D]^{-1}, \quad [M] = [D]$$

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.

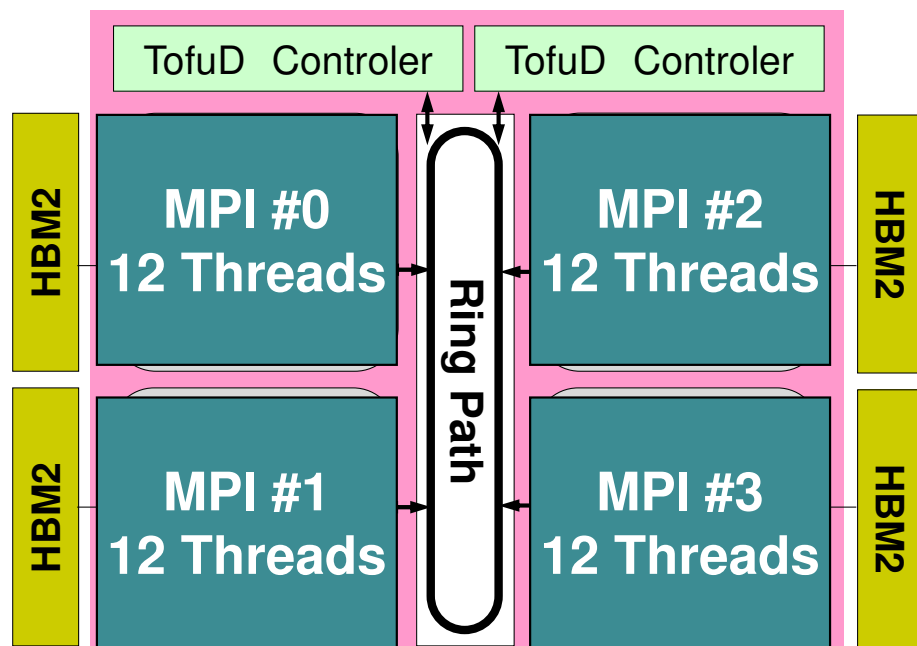
A64FX Processor on Odyssey



Name	A64FX
Processor # (Core #)	1 (48+ 2or4 Assistant Cores)
Frequency	2.2 GHz
Peak Performance	3.3792 TFLOPS
Memory Size	32 GiB
Memory Bandwidth	1,024 GB/s
L1 Cache	64 KiB/core (Inst/Data)
L2 Cache	8 MiB/CMG

- 4 CMG's (Core Memory Group), 12 cores/CMG
 - 48 Cores/Node (Processor)
 - $2.2\text{GHz} \times 32\text{DP} \times 48 = 3379.2 \text{ GFLOPS} = 3.3792 \text{ TFLOPS}$
- NUMA Architecture (Non-Uniform Memory Access)
 - Each core of a CMG can access to the memory on other CMG's
 - Utilization of the local memory is more efficient

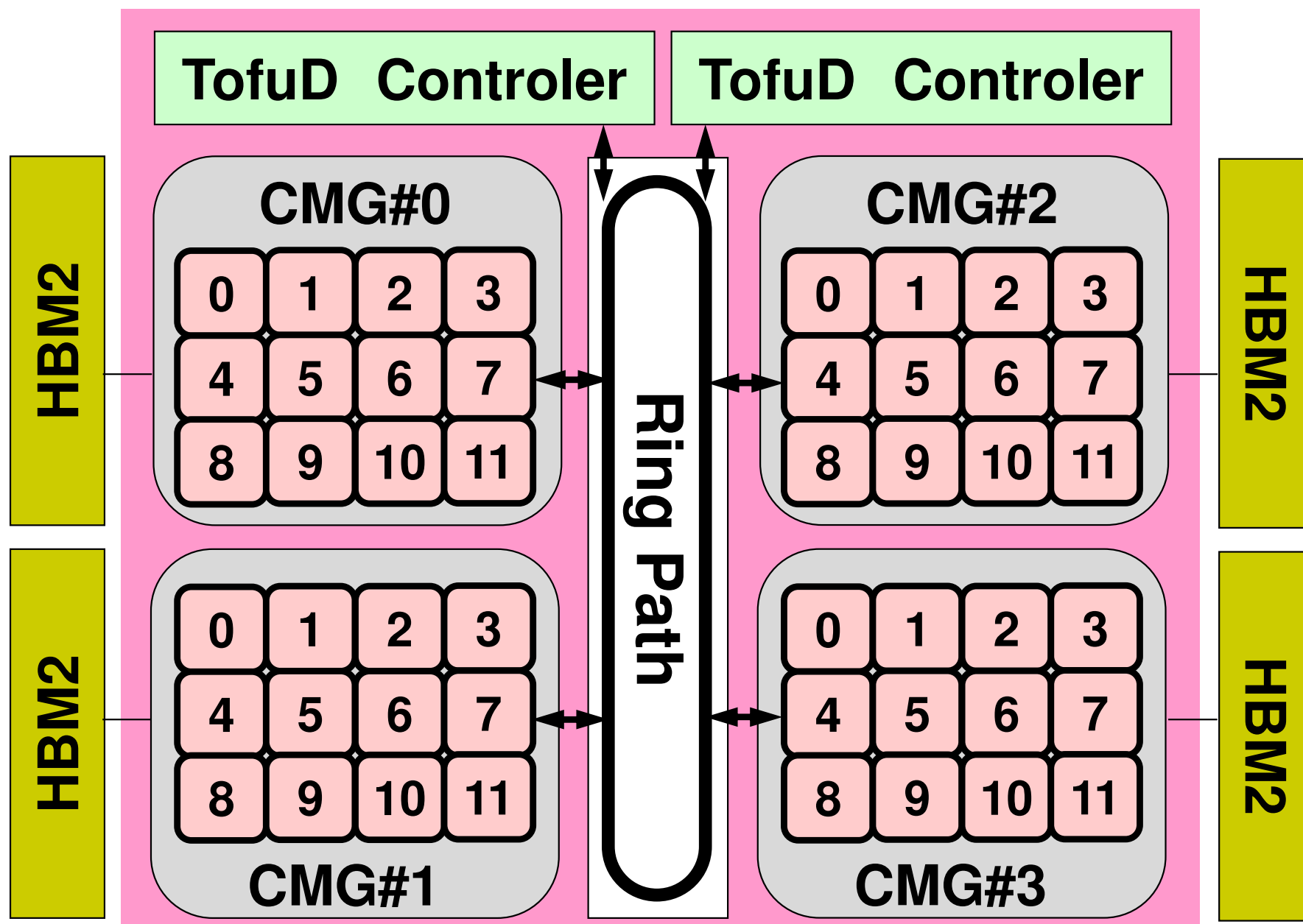
A64FX Processor on Odyssey



Name	A64FX
Processor # (Core #)	1 (48+ 2or4 Assistant Cores)
Frequency	2.2 GHz
Peak Performance	3.3792 TFLOPS
Memory Size	32 GiB
Memory Bandwidth	1,024 GB/s
L1 Cache	64 KiB/core (Inst/Data)
L2 Cache	8 MiB/CMG

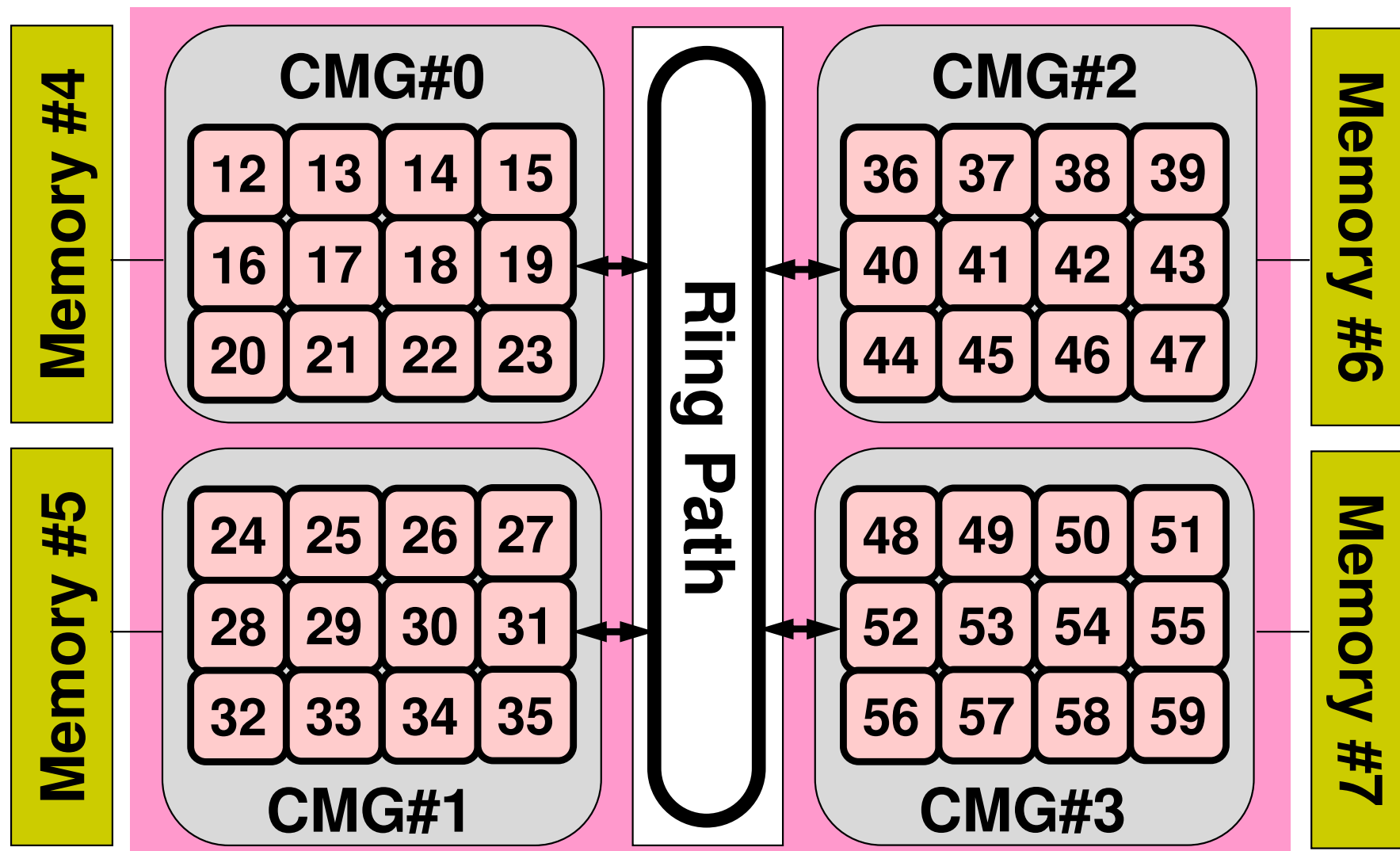
- 4 CMG's (Core Memory Group), 12 cores/CMG
 - 48 Cores/Node (Processor)
 - $2.2\text{GHz} \times 32\text{DP} \times 48 = 3379.2 \text{ GFLOPS} = 3.3792 \text{ TFLOPS}$
- NUMA Architecture (Non-Uniform Memory Access)
 - Each core of a CMG can access to the memory on other CMG's
 - Utilization of the local memory is more efficient

A64FX: CMG (Core Memory Group)



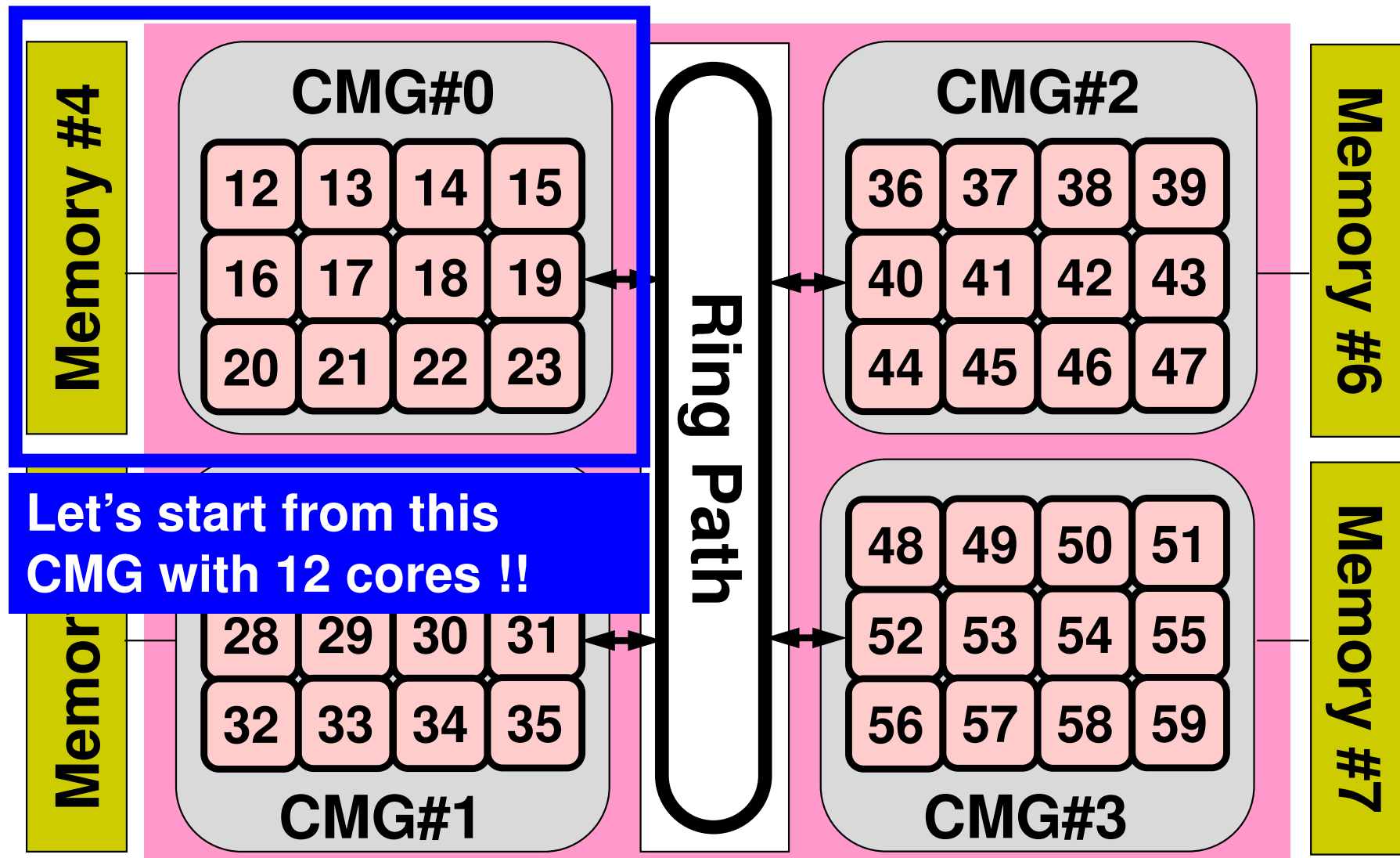
ID of CMGs, Cores, Memory's (1/2)

CMG:#0-#3, Core:#12-59, Memory:#4-#7



ID of CMGs, Cores, Memory's (1/2)

CMG:#0-#3, Core:#12-59, Memory:#4-#7



Files on Wisteria/BDEC-01 (Odyssey)

```
>$ cd /work/gt80/t80XYZ  
>$ cp /work/gt00/z30088/ompw.tar .
```

```
>$ tar xvf ompw.tar
```

```
>$ cd ompw
```

```
<$O-ompw>
```

Please make sure that following directories are created

```
run src-c0 src-c1 src-c2 src-c3 src-c3b src-f0 src-f1 src-f2 src-f3 src-f4
```

```
>$ module load fj
```

Please type this at every login !

```
>$ make -f makec
```

```
>$ ls run/solc0  
solc0
```

```
>$ cd run
```

```
<modify "INPUT.DAT", "c12.sh">
```

```
>$ pjsub c12.sh
```

<\$O-ompw>/makec

default:

```
(cd src-c0 ; make )  
(cd src-c1 ; make )  
(cd src-c2 ; make )  
(cd src-c3 ; make )  
(cd src-c3b ; make )
```

clean:

```
(cd src-c0 ; make clean)  
(cd src-c1 ; make clean)  
(cd src-c2 ; make clean)  
(cd src-c3 ; make clean)  
(cd src-c3b ; make clean)
```

<\$O-ompw>/makec-org

default:

```
(cd src-c0 ; make -f make-o)  
(cd src-c1 ; make -f make-o)  
(cd src-c2 ; make -f make-o)  
(cd src-c3 ; make -f make-o)
```

clean:

```
(cd src-c0 ; make -f make-o clean)  
(cd src-c1 ; make -f make-o clean)  
(cd src-c2 ; make -f make-o clean)  
(cd src-c3 ; make -f make-o clean)
```

<\$O-ompw>/src-c0/Makefile, make-o

parallel computing by OpenMP

Makefile

```
CC      = fccpx
OPTFLAG= -Kfast,openmp -Nclang -msve-vector-bits=512 -ffj-ocl
TARGET  = ../run/solc0

.SUFFIXES:
.SUFFIXES: .o .c

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

$(OBJS): $(HEADERS)

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

make-o

```
CC      = fccpx
OPTFLAG= -Kfast,openmp
TARGET  = ../run/solc0org

.SUFFIXES:
.SUFFIXES: .o .c

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

```

C-Compiler : 2-modes

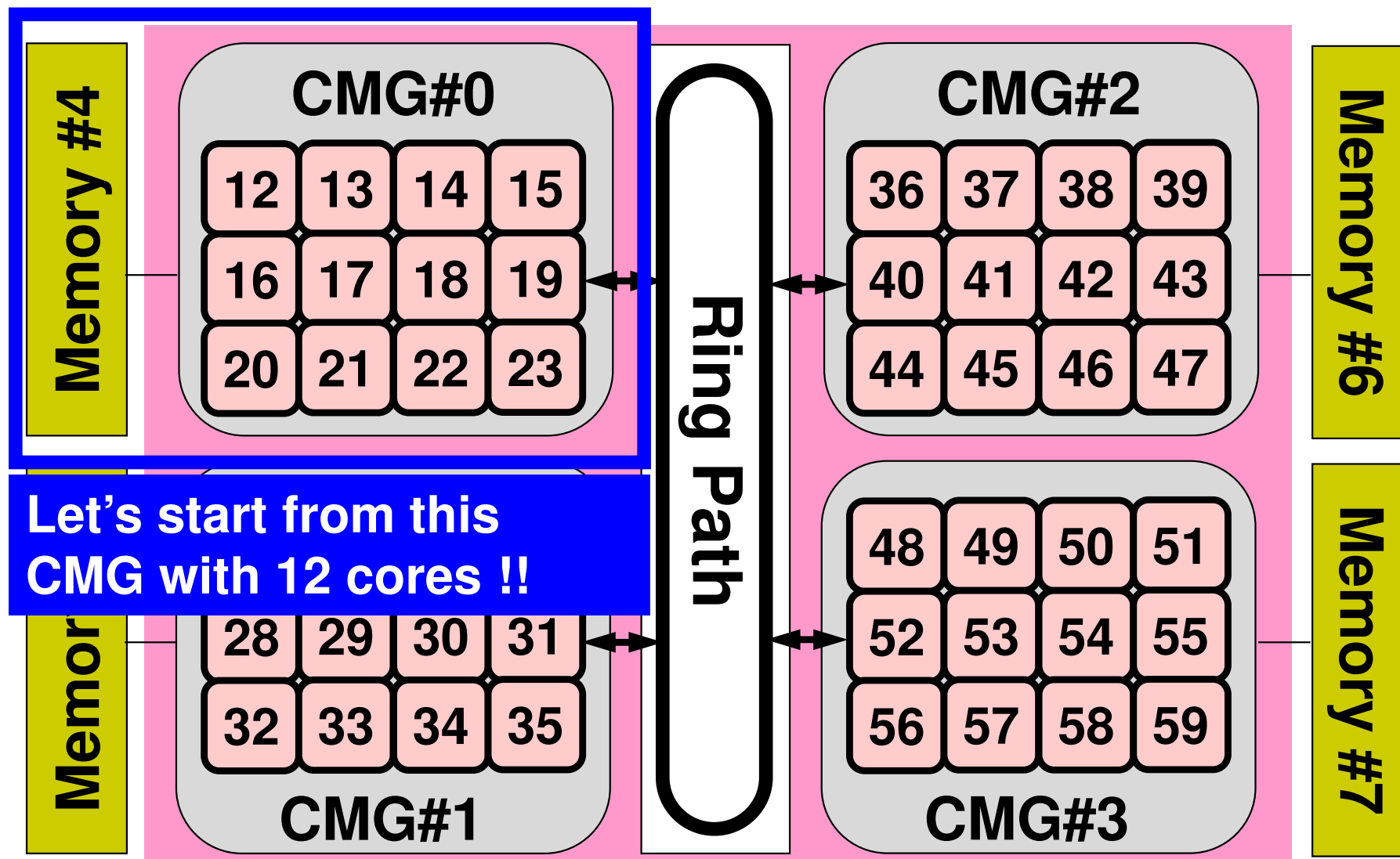
trad (-Nnoclang) (default)	• Based on Fujitsu's compiler developed for K and PRIMEHPC FX100 or older • Compatible with Fujitsu's Traditional Compilers • C89/C99/C11, OpenMP 3.1/OpenMP 4.5 (partially) • Default (-Nnoclang) • Generally slow for the materials in this class • make -f make-org (make-o)
clang (-Nclang)	• Based on Clang/LLVM Compilers (Open Source) • Suitable for using Most Updated Capability's, and for using OSS (Open Source Software) • C89/C99/C11, OpenMP 4.5/OpenMP 5.0 (partially) • Generally faster than "trad" modes, difference between "trad" and "clang" is smaller for optimized codes • In this class, default is "clang" mode • make -f makeec (Makefile)

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

ID of CMGs, Cores, Memory's (1/2)

CMG:#0-#3, Core:#12-59, Memory:#4-#7



Job Script (1/3): c12.sh

- `/work/gt80/t80XXX/ompw/run/c12.sh`
- Scheduling + Shell Script

```
#!/bin/sh
#PJM -N "c12"                               Job Name (not required)
#PJM -L rscgrp=lecture0-o                     Name of Queue (Resource Group)
#PJM -L node=1                               Node # (=1)
#PJM --omp thread=12                         Thread # (1-48, ~12 for a while)
#PJM -L elapse=00:15:00                     Elapsed Computation Time
#PJM -g gt80                                Group Name (Wallet)
#PJM -j
#PJM -e err                                 Standard Error
#PJM -o c12.lst                             Standard Output

module load fj
export OMP_NUM_THREADS=12                   Thread # (--omp thread=XX)
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

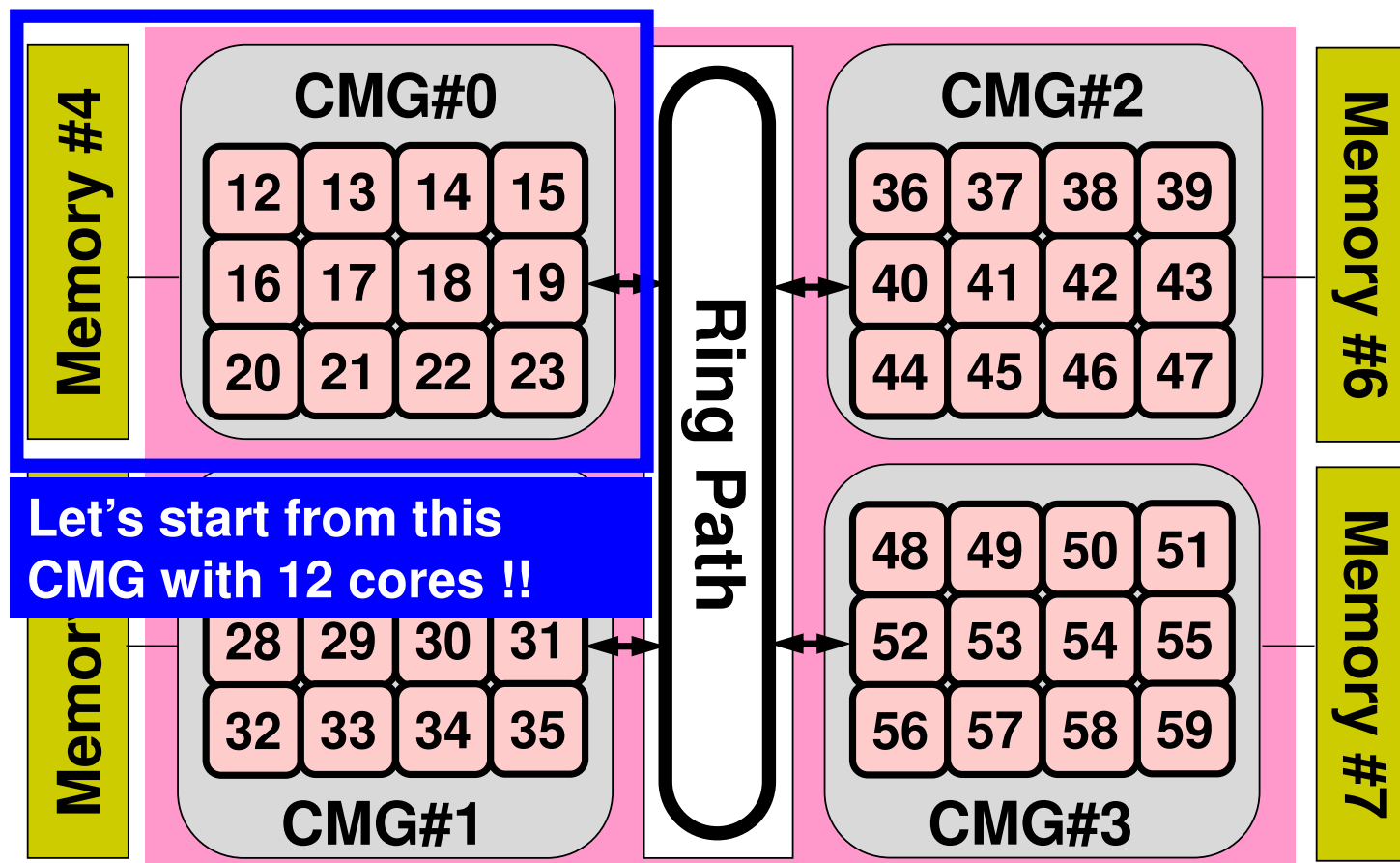
numactl ./solc0
numactl -C 12-23 -m 4 ./solc0
```

Job Script (2/3): c12.sh

- `/work/gt80/t80XXX/ompw/run/c12.sh`
- `numactl`: utilization of local resources as much as possible

```
numactl ./solc0
```

```
numactl -C 12-23 -m 4 ./solc0
```



Job Script (3/3): c12.sh

```
export XOS_MMM_L_PAGING_POLICY=
demand:demand:demand
```

Parameters	Values (Underline: Default)	Description
XOS_MMM_L_PAGING_POLICY	[demand <u>prepage</u>] [<u>demand</u> prepage] [demand <u>prepage</u>]	Paging policy (page allocation trigger) of each memory unit ✓ demand: Demand Paging Method ✓ prepage: Prepaging Method 3 Items are defined ✓ 1st Item: .bss area of static data (.data area of static data is always “prepage”) ✓ 2nd Item: Stack Area, Thread Stack Area ✓ 3rd Item: Area for Dynamic Memory Allocation If a value other than the specified value (demand/prepage), the configuration is considered as “prepage:demand:prepage” “demand:demand:demand” is recommended for using multiple CMG’s

Running Jobs

```
>$ cd /work/gt80/t80XYZ  
>$ cd ompw/run  
>$ pjsub c12.sh  
  
>$ cat c12.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	

Output (1/2) (Fortran)

```
[t00XYZ@wisteria01 run]$ cat f12.lst
```

```
128      128      128
  1      8.958216E+00
101      8.313496E+00
201      2.090443E+00
301      3.811029E-01
401      3.769653E-02
501      9.429978E-04
601      4.940783E-05
701      1.888611E-06
801      2.243179E-08
826      9.818026E-09
```

```
5.275418E+00 sec. (solver)
```

```
##ANSWER      2097152      1.459831E+04
```

```
128      128      128
  1      8.958216E+00
101      8.313496E+00
201      2.090443E+00
301      3.811029E-01
401      3.769653E-02
501      9.429978E-04
601      4.940783E-05
701      1.888611E-06
801      2.243179E-08
826      9.818026E-09
```

```
5.270398E+00 sec. (solver)
```

```
##ANSWER      2097152      1.459831E+04
```

Output (2/2) : 5-times

```
[t00XYZ@wisteria01 run]$ grep "(sol" c12.lst
```

```
5.275418E+00 sec. (solver)
5.270398E+00 sec. (solver)
5.270445E+00 sec. (solver)
5.271554E+00 sec. (solver)
5.270543E+00 sec. (solver)
```

```
numactl ./solc0
```

```
5.272427E+00 sec. (solver)
5.272081E+00 sec. (solver)
5.270522E+00 sec. (solver)
5.271067E+00 sec. (solver)
5.271237E+00 sec. (solver)
```

```
numactl -C 12-23 -m 4 ./solc0
```

Available Resource Groups (Queue's)

- Following 2 resource groups are available
- Up to 12 nodes are available, while you need a single node in this class
 - **lecture-o**
 - 12 nodes (576 cores), 15 min., valid until the end of Sep. 2022
 - Shared by all “educational” users
 - **lecture0-o**
 - 12 nodes (576 cores), 15 min., active during class time
 - More jobs (compared to **lecture-o**) 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`

```
[t00470@wisteria01 run]$ pjsub f2_48.sh
```

```
[INFO] PJM 0000 pjsub Job 15713 submitted.
```

```
[t00470@wisteria01 run]$ pjsub f3_48.sh
```

```
[INFO] PJM 0000 pjsub Job 15714 submitted.
```

```
[t00470@wisteria01 run]$ pjstat
```

```
Wisteria/BDEC-01 scheduled stop time: 2021/05/28(Fri) 09:00:00 (Remain: 4days 1:25:56)
```

JOB_ID	JOB_NAME	STATUS	PROJECT	RSCGROUP	START_DATE	ELAPSE	TOKEN	NODE	GPU
15713	f2_48	RUNNING	gt00	lecture-o	05/24 07:34:03	00:00:02	-	1	-
15714	f3_48	QUEUED	gt00	lecture-o	--/-- --:--:--	00:00:00	-	1	-

```
[t00470@wisteria01 run]$ pjstat
```

```
Wisteria/BDEC-01 scheduled stop time: 2021/05/28(Fri) 09:00:00 (Remain: 4days 1:25:56)
```

JOB_ID	JOB_NAME	STATUS	PROJECT	RSCGROUP	START_DATE	ELAPSE	TOKEN	NODE	GPU
15713	f2_48	RUNNING	gt00	lecture-o	05/24 07:34:03	00:00:02	-	1	-
15714	f3_48	RUNNING	gt00	lecture-o	(05/24 07:34)	00:00:00	-	1	-

```
[t00XYZ@wisteria01 ~]$ pjdel 15714
```

```
[INFO] PJM 0100 pjdel Accepted Job 15714
```

```
[t00XYZ@wisteria01 ~]$ pjstat
```

```
Wisteria/BDEC-01 scheduled stop time: 2021/05/28(Fri) 09:00:00 (Remain: 4days 1:25:56)
```

JOB_ID	JOB_NAME	STATUS	PROJECT	RSCGROUP	START_DATE	ELAPSE	TOKEN	NODE	GPU
15713	f2_48	RUNNING	gt00	lecture-o	05/24 07:34:03	00:00:02	-	1	-

```
[t00XYZ@wisteria01 ~]$ pjstat
```

```
Wisteria/BDEC-01 scheduled stop time: 2021/05/28(Fri) 09:00:00 (Remain: 4days 1:21:45)
```

```
No unfinished job found.
```

```
[t00XYZ@wisteria01 ~]$ pjstat --rsc
```

```
SYSTEM: Odyssey
```

RSCGRP	STATUS	NODE
lecture-o	[ENABLE, START]	96
lecture0-o	[DISABLE, STOP]	2x12x16

```
[t00XYZ@wisteria01 ~]$ pjstat --rsc -x
```

```
SYSTEM: Odyssey
```

RSCGRP	STATUS	MIN_NODE	MAX_NODE	MAX_ELAPSE	REMAIN_ELAPSE	MEM(GiB)	PROJECT
lecture-o	[ENABLE, START]	1	12	00:15:00	00:15:00	28	gt00
lecture0-o	[DISABLE, STOP]	1	12	00:15:00	--:--:--	28	gt00

```
[t00XYZ@wisteria01 ~]$ pjstat --limit
```

```
SYSTEM: Odyssey
```

PROJECT	ACCEPT	RUN	BULK_ACCEPT	BULK_RUN	NODE
gt80	0/ 128	0/ 16	0/ 8	0/ 16	0/ 2304

```
SYSTEM: Aquarius
```

PROJECT	ACCEPT	RUN	BULK_ACCEPT	BULK_RUN	GPU
gt80	0/ 4	0/ 2	0/ 0	0/ 0	0/ 64

poi_gen (1/2): Main Part

Be carefule with “private” !!

```

#pragma omp parallel for private
(ice1, icN1, icN2, icN3, icN4, icN5, icN6, VOL0, isLU, icou, coef, ii, jj, kk)

    for(ice1=0; ice1<ICELTOT; ice1++) {
        icN1 = NEIBcell[ice1][0];
        icN2 = NEIBcell[ice1][1];
        icN3 = NEIBcell[ice1][2];
        icN4 = NEIBcell[ice1][3];
        icN5 = NEIBcell[ice1][4];
        icN6 = NEIBcell[ice1][5];

        VOL0 = VOLCEL[ice1];
        isLU=indexLU[ice1];
        icou= 0;
        if(icN5 != 0) {
            coef = RDZ * ZAREA;
            D[ice1] -= coef;
            itemLU[icou+isLU]= icN5-1;
            AMAT [icou+isLU]= coef;
            icou= icou + 1;
        }

        (...)

        ii = XYZ[ice1][0];
        jj = XYZ[ice1][1];
        kk = XYZ[ice1][2];

        BFORCE[ice1] = - (double)(ii + jj + kk) * VOLCEL[ice1];
    }

```

poi_gen (2/2): Boundary Conditions

Be carefule with “private” !!

```
#pragma omp parallel for private (ib, icel, coef)
```

```
    for (ib=0; ib<ZmaxCELtot; ib++) {  
        icel  = ZmaxCEL[ib];  
        coef = 2.0 * RDZ * ZAREA;  
        D[icel-1] -= coef;  
    }
```

solve_PCG (1/5)

parallel computing by OpenMP

Interface has been changed

restrict: pointers are not referred from
other arrays

```
#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 *restrict indexLU, int *restrict itemLU,
          double *restrict D, double *restrict B, double *restrict X,
          double *restrict AMAT, double EPS, int *restrict ITR, int *restrict IER)
{
```

```
    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;
```

```
    double (*restrict W)[N] = (double (*)[N])malloc(4*sizeof(double[N]));
    if(W == NULL) {
        fprintf(stderr, "Error: %s\n", strerror(errno));
        return -1;
    }
```

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

#pragma omp parallel for private (i, VAL, j)
  for(i=0; i<N; i++) {
    VAL = D[i] * X[i];
    for(j=indexLU[i]; j<indexLU[i+1]; j++) {
      VAL += AMAT[j] * X[itemLU[j]];
    }
    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];
  }

```

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) = 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, ...
    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=indexLU[i]; j<indexLU[i+1]; j++) {
        VAL += AMAT[j] * W[P][itemLU[j]];
    }
    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 $i = 1, 2, \dots$

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

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

c01,c02,c04,c06,c08,c12.sh

- /work/gt80/t80XXX/ompw/run/cXY.sh
- Scheduling + Shell Script

```
#!/bin/sh
#PJM -N "cYZ"                Job Name (not required)
#PJM -L rscgrp=lecture0-o    Name of Queue (Resource Group)
#PJM -L node=1              Node # (=1)
#PJM --omp thread=YZ        Thread # (1-48, ~12 for a while)
#PJM -L elapse=00:15:00     Elapsed Computation Time
#PJM -g gt80                Group Name (Wallet)
#PJM -j
#PJM -e err                 Standard Error
#PJM -o cYZ.lst             Standard Output

module load fj
export OMP_NUM_THREADS=YZ    Thread # (--omp thread=YZ)
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl ./solc0
numactl -C 12-23 -m 4 ./solc0
```

Time for PCG: Etime-Stime: Fortran

$NX=NY=NZ=128$

5 measurements, best result

Efficiency decreases as core# increases: decreasing of memory performance

Large
大



Granularity: 粒度
Problem Size/Thread

Small
小



Thread #	sec	Speed-up	Parallel Efficiency (%)
1	50.27	1.00	100.00
2	25.24	1.99	99.60
4	12.98	3.87	96.86
6	9.24	5.44	90.73
8	7.27	6.92	86.50
12	5.09	9.88	82.30

Parallel Efficiency(%)= 100*(Speed-Up)/Thread#

c04.sh

```
#!/bin/sh
#PJM -N "c04"
#PJM -L rscgrp=lecture-o
#PJM -L node=1
#PJM --omp thread=4
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o c04.lst

module load fj
export OMP_NUM_THREADS=4
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl ./solc0
numactl ./solc0
numactl ./solc0
numactl ./solc0
numactl ./solc0
numactl -C 12-23 -m 4 ./solc0
numactl -C 12-23 -m 4 ./solc0
numactl -C 12-23 -m 4 ./solc0
numactl -C 12-23 -m 4 ./solc0
numactl -C 12-23 -m 4 ./solc0
```

c08.sh

```
#!/bin/sh
#PJM -N "c08"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=8
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o c08.lst

module load fj
export OMP_NUM_THREADS=8
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl ./solc0
numactl ./solc0
numactl ./solc0
numactl ./solc0
numactl ./solc0
numactl -C 12-23 -m 4 ./solc0
numactl -C 12-23 -m 4 ./solc0
numactl -C 12-23 -m 4 ./solc0
numactl -C 12-23 -m 4 ./solc0
numactl -C 12-23 -m 4 ./solc0
```

Multiple CMG's

Time for PCG: Etime-Stime: Fortran

$NX=NY=NZ=128$

5 measurements, best result

Large
大



Granularity: 粒度
Problem Size/Thread

Small
小

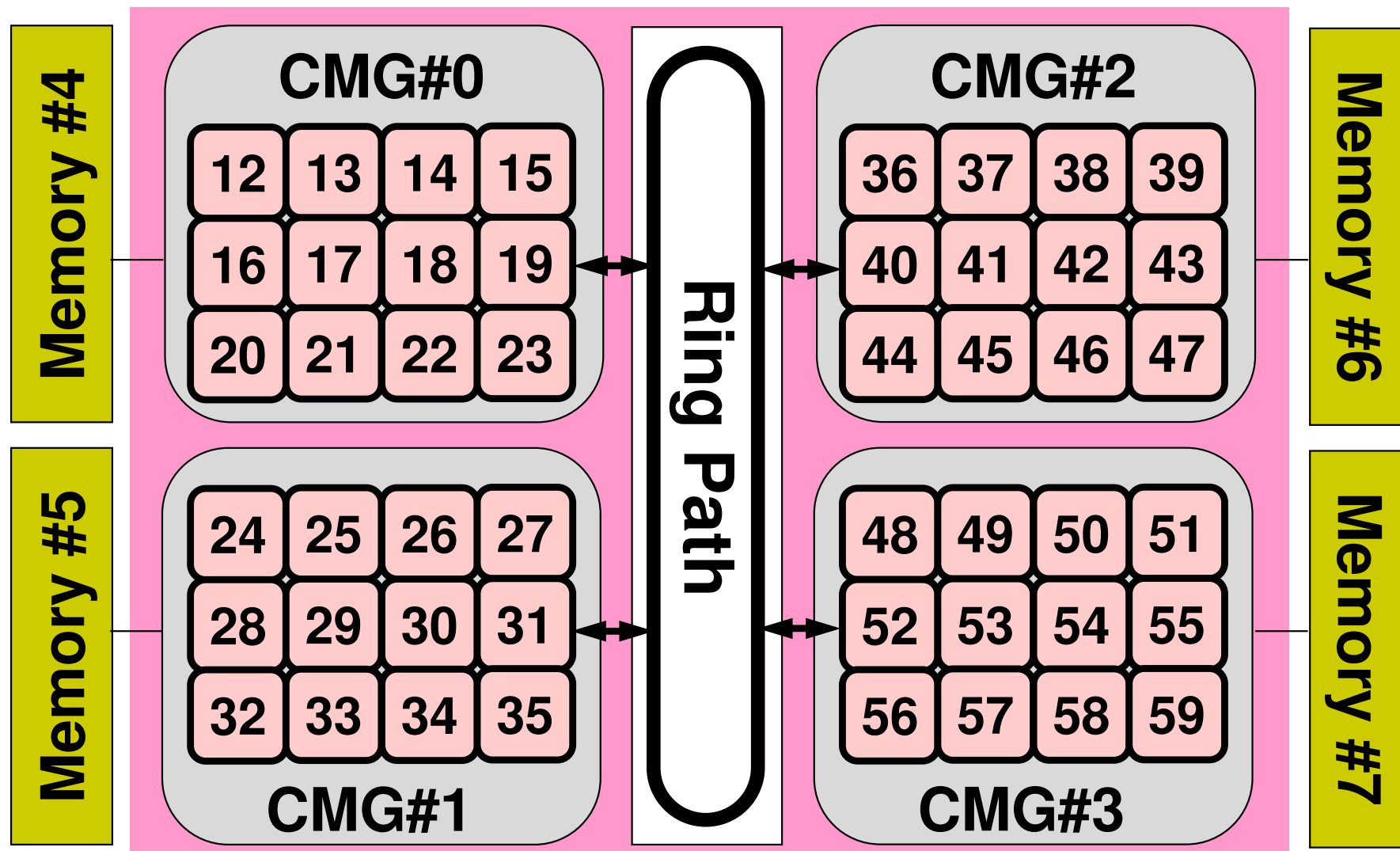


Thread #	sec	Speed-up	Parallel Efficiency (%)
12	5.09	12.00	100.00
24	2.79	21.88	91.18
36	1.99	30.75	85.41
48	1.70	35.97	74.95

Parallel Efficiency(%)= 100*(Speed-Up)/Thread#

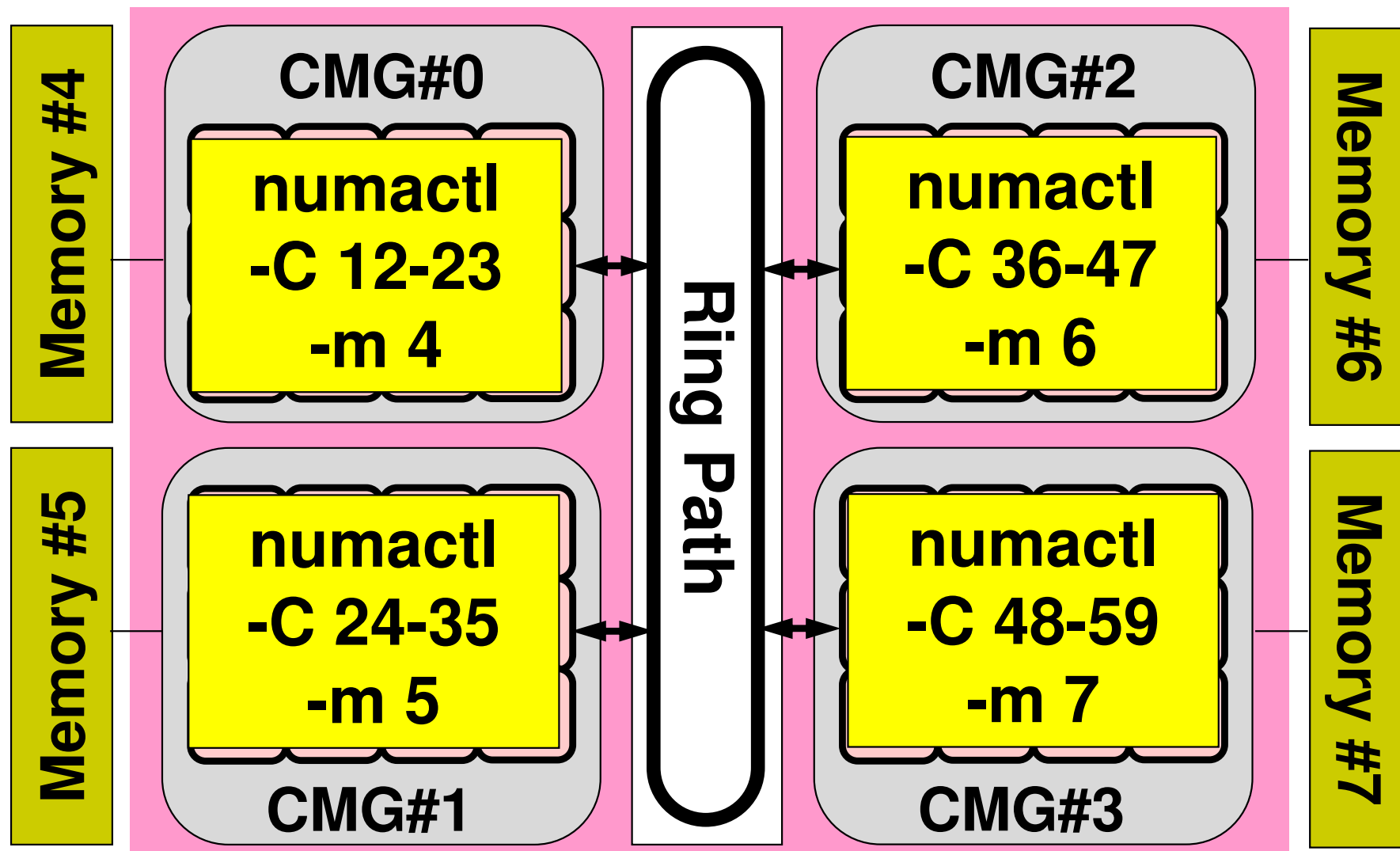
ID of CMGs, Cores, Memory's (1/2)

CMG:#0-#3, Core:#12-59, Memory:#4-#7



ID of CMGs, Cores, Memory's (2/2)

CMG:#0-#3, Core:#12-59, Memory:#4-#7



c0_12.sh

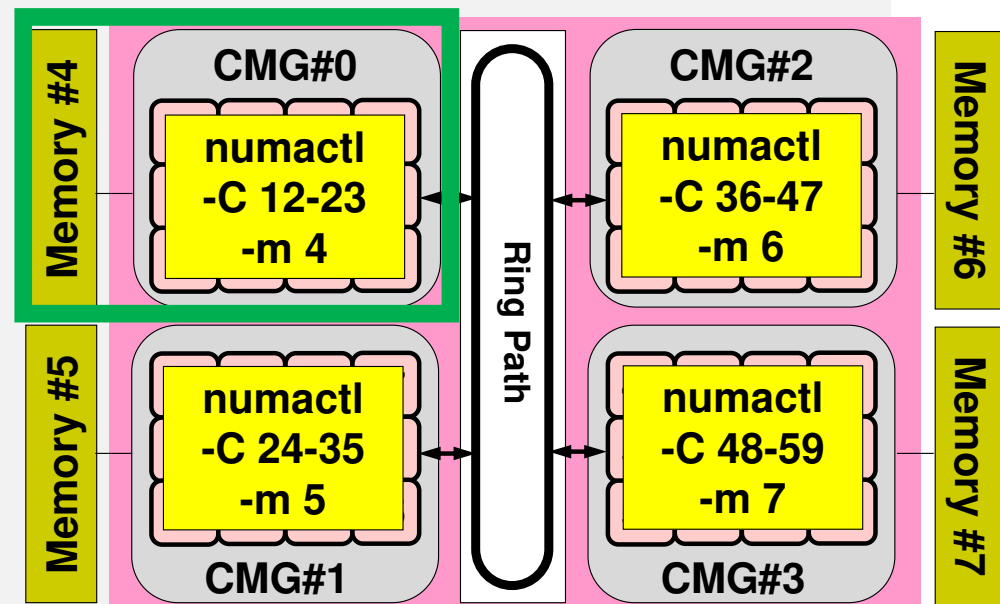
```
#!/bin/sh
#PJM -N "c0_12"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=12
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o c0_12.1st
```

```
module load fj
```

```
export OMP_NUM_THREADS=12
```

```
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand
```

```
numactl ./solc0
numactl ./solc0
numactl ./solc0
numactl ./solc0
numactl ./solc0
numactl -C 12-23 -m 4 ./solc0
numactl -C 12-23 -m 4 ./solc0
numactl -C 12-23 -m 4 ./solc0
numactl -C 12-23 -m 4 ./solc0
numactl -C 12-23 -m 4 ./solc0
```



c0_24.sh

```
#!/bin/sh
#PJM -N "c0_24"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=24
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o c0_24.1st
```

```
module load fj
```

```
export OMP_NUM_THREADS=24
```

```
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand
```

```
numactl ./solc0
```

```
numactl ./solc0
```

```
numactl ./solc0
```

```
numactl ./solc0
```

```
numactl ./solc0
```

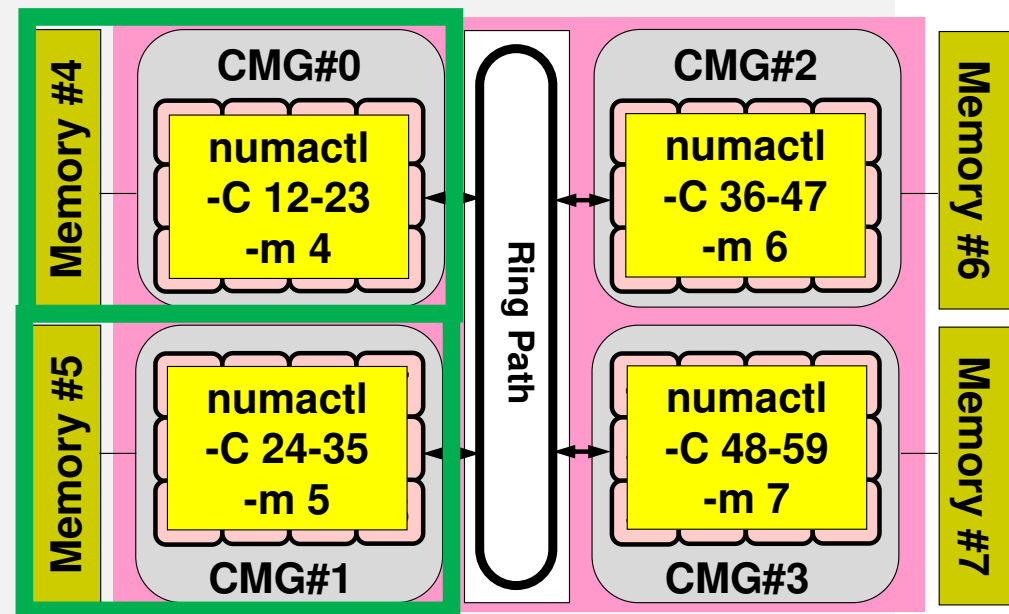
```
numactl -C 12-35 -m 4-5 ./solc0
```

```
numactl -C 12-35 -m 4-5 ./solc0
```

```
numactl -C 12-35 -m 4-5 ./solc0
```

```
numactl -C 12-35 -m 4-5 ./solc0
```

```
numactl -C 12-35 -m 4-5 ./solc0
```



c0_36.sh

```
#!/bin/sh
#PJM -N "c0_36"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=36
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o c0_36.1st
```

```
module load fj
```

```
export OMP_NUM_THREADS=36
```

```
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand
```

```
numactl ./solc0
```

```
numactl ./solc0
```

```
numactl ./solc0
```

```
numactl ./solc0
```

```
numactl ./solc0
```

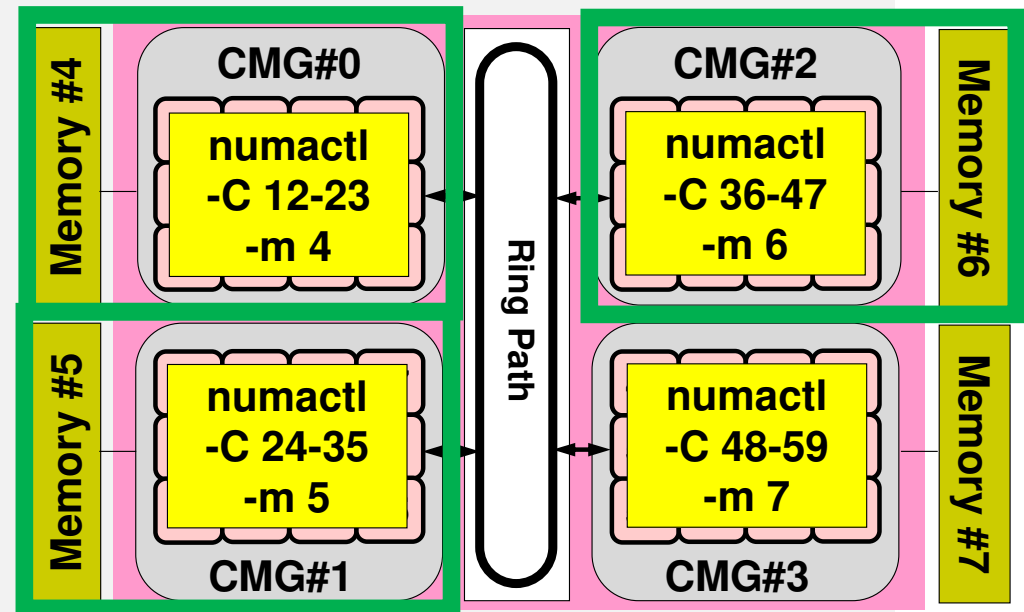
```
numactl -C 12-47 -m 4-6 ./solc0
```

```
numactl -C 12-47 -m 4-6 ./solc0
```

```
numactl -C 12-47 -m 4-6 ./solc0
```

```
numactl -C 12-47 -m 4-6 ./solc0
```

```
numactl -C 12-47 -m 4-6 ./solc0
```



c0_48.sh

```
#!/bin/sh
#PJM -N "c0_48"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=48
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o c0_48.1st
```

```
module load fj
```

```
export OMP_NUM_THREADS=48
```

```
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand
```

```
numactl ./solc0
```

```
numactl ./solc0
```

```
numactl ./solc0
```

```
numactl ./solc0
```

```
numactl ./solc0
```

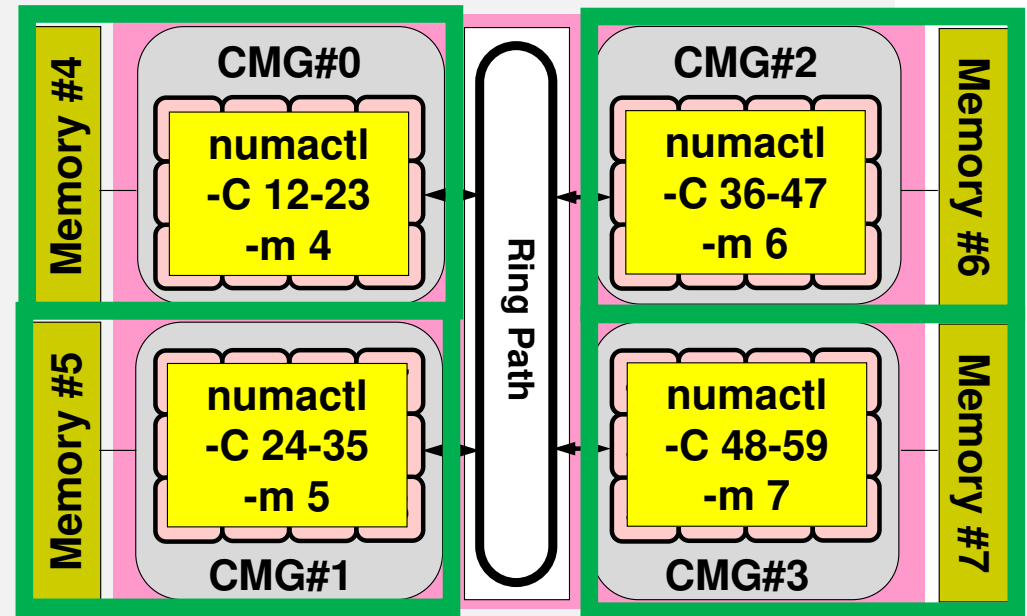
```
numactl -C 12-59 -m 4-7 ./solc0
```

```
numactl -C 12-59 -m 4-7 ./solc0
```

```
numactl -C 12-59 -m 4-7 ./solc0
```

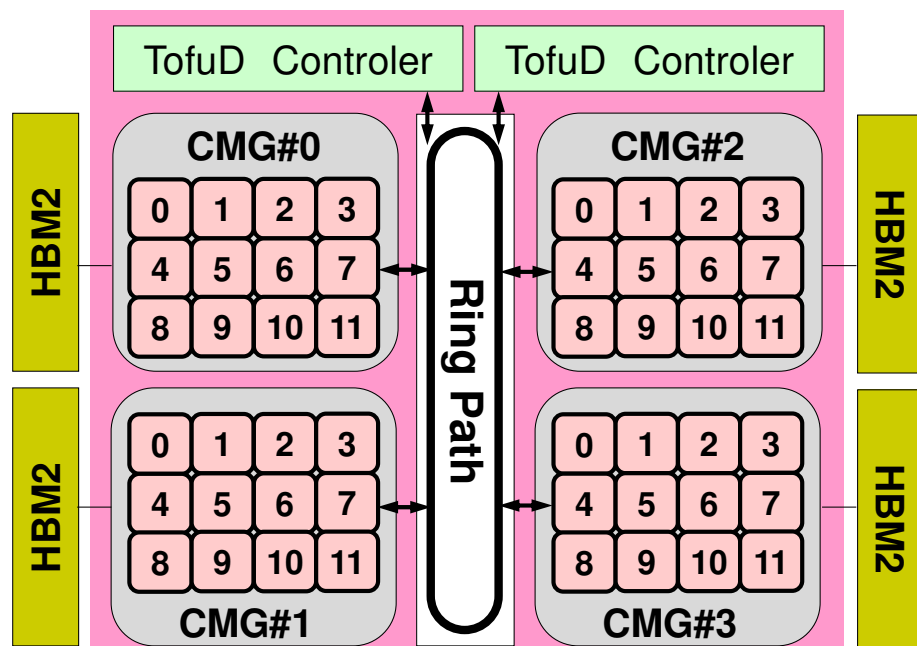
```
numactl -C 12-59 -m 4-7 ./solc0
```

```
numactl -C 12-59 -m 4-7 ./solc0
```



- OpenMP
- Login to Wisteria/BDEC-01
- Parallel Code by OpenMP (0): up to 12 cores
- **Parallel Code by OpenMP (1): First Touch**
- Parallel Code by OpenMP (2): +ELL
- Parallel Code by OpenMP (3): reduced omp-parallel
- Parallel Code by OpenMP (4): Further Optimization (Fortran only)

A64FX Processor on Odyssey



Name	A64FX
Processor # (Core #)	1 (48+ 2or4 Assistant Cores)
Frequency	2.2 GHz
Peak Performance	3.3792 TFLOPS
Memory Size	32 GiB
Memory Bandwidth	1,024 GB/s
L1 Cache	64 KiB/core (Inst/Data)
L2 Cache	8 MiB/CMG

- 4 CMG's (Core Memory Group), 12 cores/CMG
 - 48 Cores/Node (Processor)
 - $2.2\text{GHz} \times 32\text{DP} \times 48 = 3379.2 \text{ GFLOPS} = 3.3792 \text{ TFLOPS}$
- **NUMA Architecture (Non-Uniform Memory Access)**
 - Each core of a CMG can access to the memory on other CMG's
 - Utilization of the local memory is more efficient

First Touch Data Placement

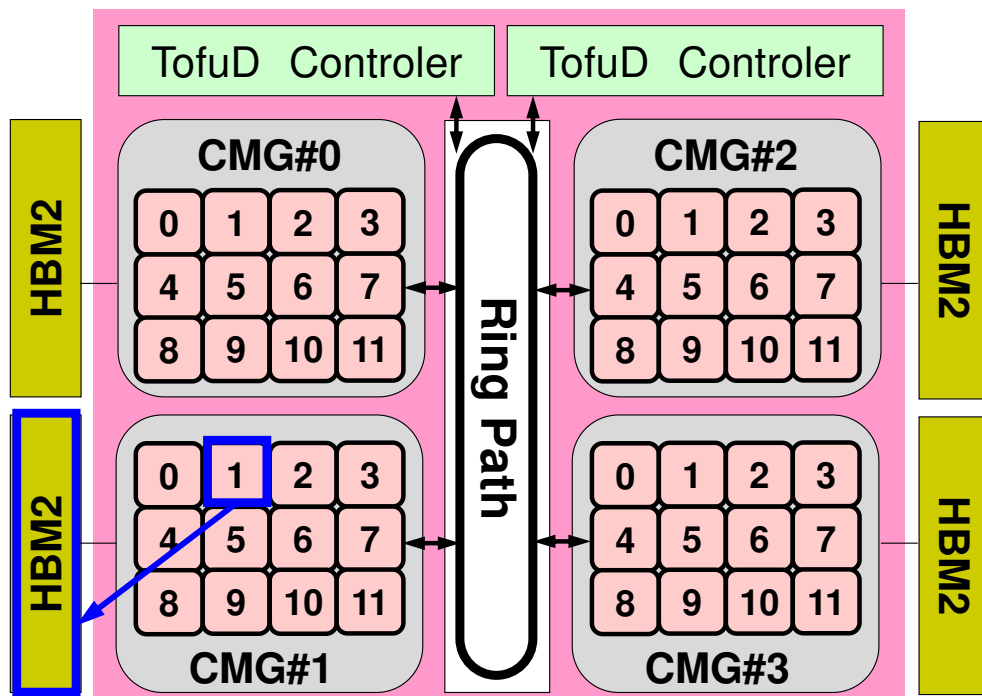
“Patterns for Parallel Programming” Mattson, T.G. et al.

- To reduce memory traffic in the system, it is important to keep the data close to the PEs that will work with the data (e.g. NUMA control).
- On NUMA computers, this corresponds to making sure the pages of memory are allocated and “owned” by the PEs that will be working with the data contained in the page.
 - ✓ Page/Memory Page/Virtual Page: A fixed-length continuous block of virtual memory, smallest unit of data for memory management in a virtual memory OS
- The most common NUMA page-placement algorithm is the “first touch” algorithm, in which the PE first referencing a region of memory will have the page holding that memory assigned to it.
- A very common technique in OpenMP program for optimization is to initialize data in parallel using the same loop schedule as will be used later in the computations.

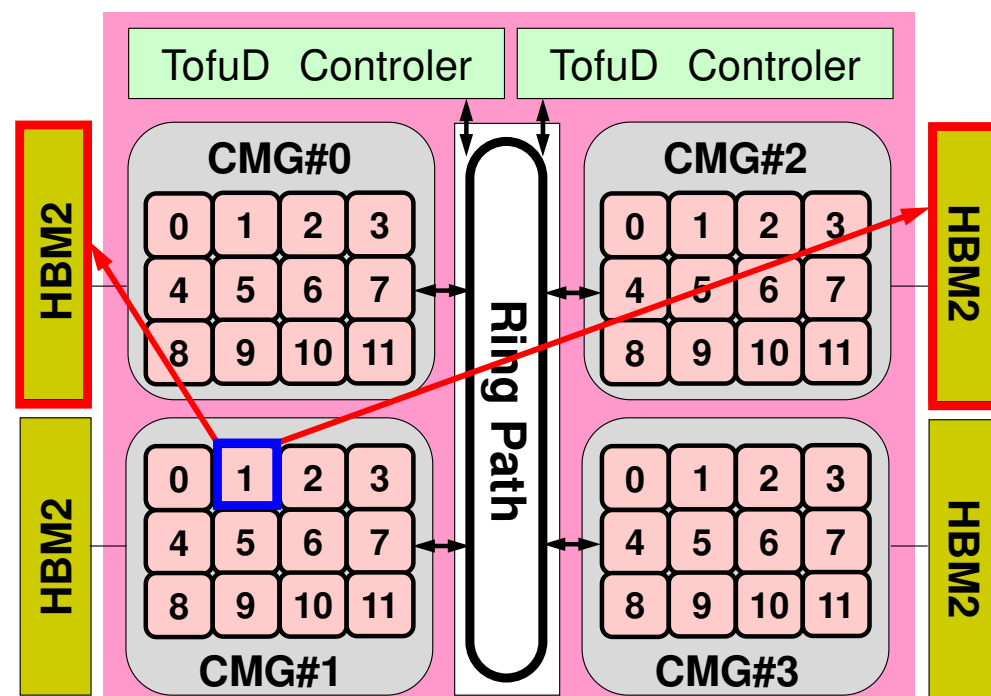
Summary: First Touch Data Placement

- On NUMA architecture (Non-Uniform Memory Access), “pages of memory” are not allocated when variables and arrays are declared/allocated in the program.
- “Pages” are allocated at the local memory of the “socket” for the “core/thread” that first touches the variables and/or arrays.
- If the pages are not on the local memory of the socket for each thread, performance of the program is very bad.
- A very common technique in OpenMP program for optimization is to initialize data in parallel using the same loop schedule as will be used later in the computations.
- You have to consider this if you use 4 CMG's of the Odyssey system for a single OpenMP program
 - If you don't care, all pages are created at the local memory of CMG#0
 - Not needed for a single CMG case

Local/Remote Memory



Local Memory



Remote Memory

Program by “First-Touch”: src-c1

Original Program in src-c0

```
>$ cd /work/gt80/t80XYZ/ompw
```

```
>$ cd run
```

```
<modify "INPUT.DAT", "c1_XY.sh"> (XY:12,24,36,48)
```

```
>$ pjsub c1_XY.sh
```

```
[XYZ@wisteria01 run]$ cd ../src-c0  
[XYZ@wisteria01 src-f0]$ diff poi_gen.c ../src-c1/poi_gen.c  
25,29c25,31
```

```
    for (i = 0; i < ICELTOT ; i++) {  
        BFORCE[i]=0.0;  
        D[i]    =0.0;  
        PHI[i]=0.0;  
        INLU[i] = 0;  
    }
```

src-c0

```
---  
#pragma omp parallel for private (i)  
    for (i = 0; i < ICELTOT ; i++) {  
        BFORCE[i]=0.0;  
        D[i]    =0.0;  
        PHI[i]=0.0;  
        INLU[i] = 0;  
    }
```

src-c1

```
for (i = 0; i <=ICELTOT ; i++) {  
    indexLU[i] = 0;  
}
```

src-c0

```
---  
#pragma omp parallel for private (i)  
for (i = 0; i <=ICELTOT ; i++) {  
    indexLU[i] = 0;  
}
```

src-c1

```
for(i=0; i<ICELTOT; i++) {  
    for(j=indexLU[i]; j<indexLU[i+1]; j++) {  
        itemLU[j]=0;  
        AMAT[j]=0.0;  
    }  
}
```

src-c0

```
---  
#pragma omp parallel for private (i,j)  
for(i=0; i<ICELTOT; i++) {  
    for(j=indexLU[i]; j<indexLU[i+1]; j++) {  
        itemLU[j]=0;  
        AMAT[j]=0.0;  
    }  
}
```

src-c1

c1_48.sh

```
#!/bin/sh
#PJM -N "c1_48"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=48
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o c1_48.lst
```

```
module load fj
```

```
export OMP_NUM_THREADS=48
```

```
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand
```

```
numactl ./solc1
```

```
numactl ./solc1
```

```
numactl ./solc1
```

```
numactl ./solc1
```

```
numactl ./solc1
```

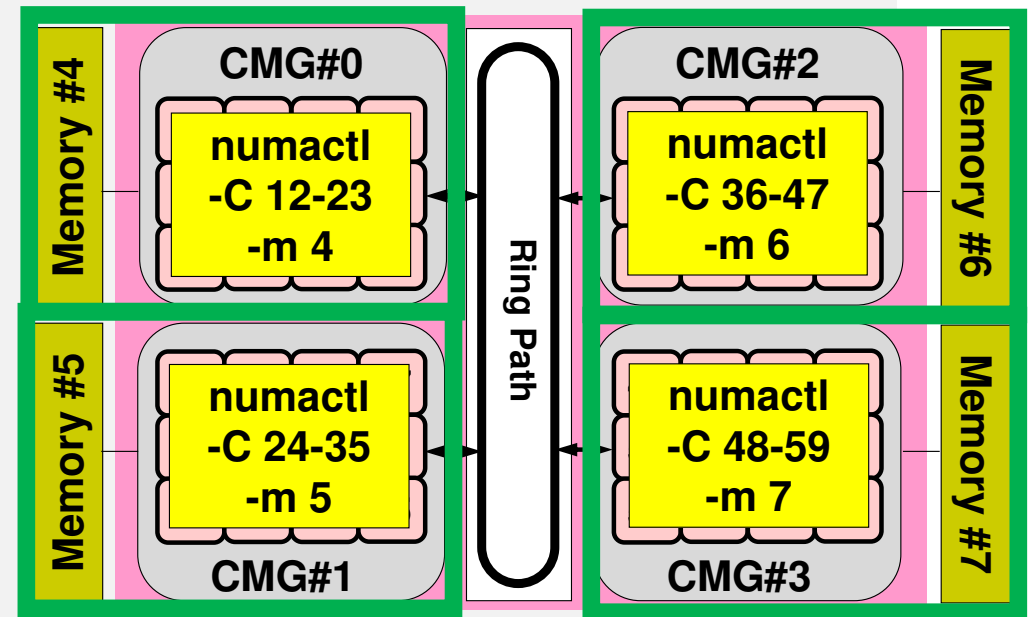
```
numactl -C 12-59 -m 4-7 ./solc1
```

```
numactl -C 12-59 -m 4-7 ./solc1
```

```
numactl -C 12-59 -m 4-7 ./solc1
```

```
numactl -C 12-59 -m 4-7 ./solc1
```

```
numactl -C 12-59 -m 4-7 ./solc1
```



Multiple CMG's

Time for PCG: Etime-Stime: Fortran

$NX=NY=NZ=128$, Best Case for 5 Measurements
based on src-f0 with 12-threads

src-f0	Thread #	sec	Speed-up	Parallel Efficiency (%)
	12	5.09	12.00	100.00
	24	2.79	21.88	91.18
	36	1.99	30.75	85.41
	48	1.70	35.97	74.95
src-f1	12	5.27	-	-
	24	2.70	22.61	94.20
	36	1.87	32.65	90.69
	48	1.52	40.09	83.51

Multiple CMG's

Time for PCG: Etime-Stime:C(clang)

NX=NY=NZ=128, Best Case for 5 Measurements
based on src-c0 with 12-threads

src-c0	Thread #	sec	Speed-up	Parallel Efficiency (%)
	12	5.03	12.00	100.00
	24	2.69	22.42	93.41
	36	1.91	31.57	87.69
	48	1.57	38.51	80.23

src-c1	12	5.22	-	-
	24	2.70	22.39	93.29
	36	1.85	32.66	90.71
	48	1.17	51.70	107.7

Cache is well-utilized, because the problem size is small

Multiple CMG's

Time for PCG: Etime-Stime

$NX=NY=NZ=128$, Best Case for 5 Measurements

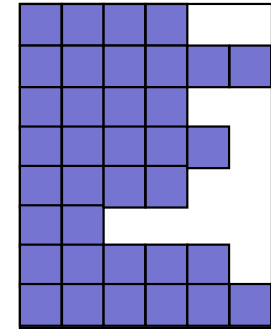
Original	Language	12	24	36	48
	Fortran	5.09	2.79	1.99	1.70
	C (clang)	5.03	2.69	1.91	1.57
	C (trad)	7.75	4.19	2.90	2.36
First-Touch	Fortran	5.27	2.70	1.87	1.52
	C (clang)	5.22	2.70	1.85	1.17
	C (trad)	7.85	4.05	2.79	1.72 (112.6 %)

- OpenMP
- Login to Wisteria/BDEC-01
- Parallel Code by OpenMP (0): up to 12 cores
- Parallel Code by OpenMP (1): First Touch
- **Parallel Code by OpenMP (2): +ELL**
- Parallel Code by OpenMP (3): reduced omp-parallel
- Parallel Code by OpenMP (4): Further Optimization (Fortran only)

Storage of Sparse Matrices

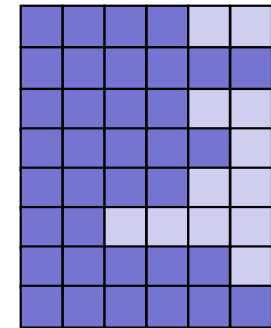
CRS (Compressed Row Storage)

```
for (i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for (j=indexLU[i]; j<indexLU[i+1]; j++) {
        VAL += AMAT[j] * W[P][itemLU[j]];
    }
    W[Q][i] = VAL;}
```



ELL (ELLPACK/ITPACK)

```
for (i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for (j=0; j<6; j++) {
        VAL += AMAT[6*i+j] * W[P][itemLU[6*i+j]];
    }
    W[Q][i] = VAL;}
```



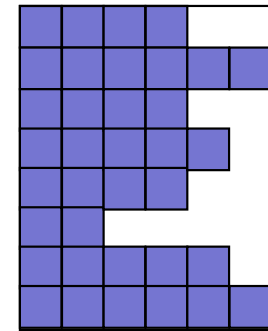
- CRS: Compressed Row Storage
 - Only non-zero off-diag's: saving memory, low performance
- ELL: ELLPACK/ITPACK
 - Fixed # of non-zero off-diag's, 0 padding needed
 - More expensive, better memory performance due to prefetch

Storage of Sparse Matrices

CRS (Compressed Row Storage)

```
for (i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for (j=indexLU[i]; j<indexLU[i+1]; j++) {
        VAL += AMAT[j] * W[P][itemLU[j]];
    }
    W[Q][i] = VAL;}

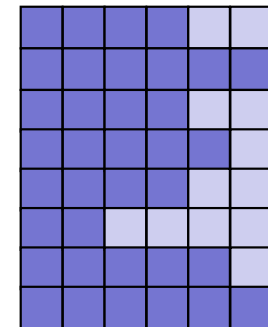
```



ELL (ELLPACK/ITPACK)

```
for (i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for (j=0; j<6; j++) {
        VAL += AMAT[6*i+j] * W[P][itemLU[6*i+j]];
    }
    W[Q][i] = VAL;}

```



ELL (ELLPACK/ITPACK) This implementation is much slower

```
for (i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for (j=0; j<6; j++) {
        VAL += AMAT[i][j] * W[P][itemLU[i][j]];
    }
    W[Q][i] = VAL;}

```

poi_gen (1/2): Private

```

#pragma omp parallel for private
(ice1, icN1, icN2, icN3, icN4, icN5, icN6, VOL0, icou, coef, ii, jj, kk)

    for(ice1=0; ice1<ICELTOT; ice1++) {
        icN1 = NEIBcell[ice1][0];
        icN2 = NEIBcell[ice1][1];
        icN3 = NEIBcell[ice1][2];
        icN4 = NEIBcell[ice1][3];
        icN5 = NEIBcell[ice1][4];
        icN6 = NEIBcell[ice1][5];

        VOL0 = VOLCEL[ice1];
        icou= 0;
        if(icN5 != 0) {
            coef = RDZ * ZAREA;
            D[ice1] -= coef;
            itemLU[6*ice1+icou]= icN5-1;
            AMAT [6*ice1+icou]= coef;
            icou= icou + 1;
        }
        if(icN3 != 0) {
            coef = RDZ * YAREA;
            D[ice1] -= coef;
            itemLU[6*ice1+icou]= icN3-1;
            AMAT [6*ice1+icou]= coef;
            icou= icou + 1;
        }
        (...)
    }

```

poi_gen (2/2): Padding

N2= 128 (poi_gen.c)

```

/*****
 * PADDING *
 *****/
    icou= 0;
    for (i = 0; i < ICELTOT ; i++) {
        for (k = 0; k < 6 ; k++) {
            if (itemLU[6*i+k]==-1) {
                icou= icou + 1;
                itemLU[6*i+k]= ICELTOT-1 + icou;
                if (icou==N2){icou=0;}
            }
        }
    }

```

N=ICELTOT				N	N+1
			N+2	N+3	N+4
		N+5	N+6	N+7	N+8
			N+9	N+10	N+11
		N+12	N+13	N+14	N+15
		N+16	N+17	N+18	N+19



			N+125	N+126	N+127
		N	N+1	N+2	N+3
		N+4	N+5	N+6	N+7
			N+8	N+9	N+10

solve_PCG (1/2)

```

W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
    fprintf(stderr, "Error: %s\n", strerror(errno));
    return -1;
}
for(i=0; i<4; i++) {
    W[i] = (double *)malloc(sizeof(double)*(N+N2));
    if(W[i] == NULL) {
        fprintf(stderr, "Error: %s\n", strerror(errno));
        return -1;
    }
}

```

```

#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)

```

```

for(i=0; i<N; i++) {
    W[DD][i] = 1.0 / D[i];
}

```

solve_PCG (2/2)

```
/******  
 * {q} = [A] {p} *  
*****/  
  
#pragma omp parallel for private (i, VAL, j)  
    for (i=0; i<N; i++) {  
        VAL = D[i] * W[P][i];  
        for (j=0; j<6; j++) {  
            VAL += AMAT[6*i+j] * W[P][itemLU[6*i+j]];  
        }  
        W[Q][i] = VAL;  
    }
```


Program by “ELL”: src-c2

```
>$ cd /work/gt80/t80XYZ/ompw
```

```
>$ cd run
```

```
<modify "INPUT.DAT", "c2_48.sh">
```

```
>$ pjsub c2_48.sh
```

c2_48.sh

```
#!/bin/sh
#PJM -N "c2_48"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=48
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o c2_48.1st
```

```
module load fj
```

```
export OMP_NUM_THREADS=48
```

```
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand
```

```
numactl ./solc2
```

```
numactl ./solc2
```

```
numactl ./solc2
```

```
numactl ./solc2
```

```
numactl ./solc2
```

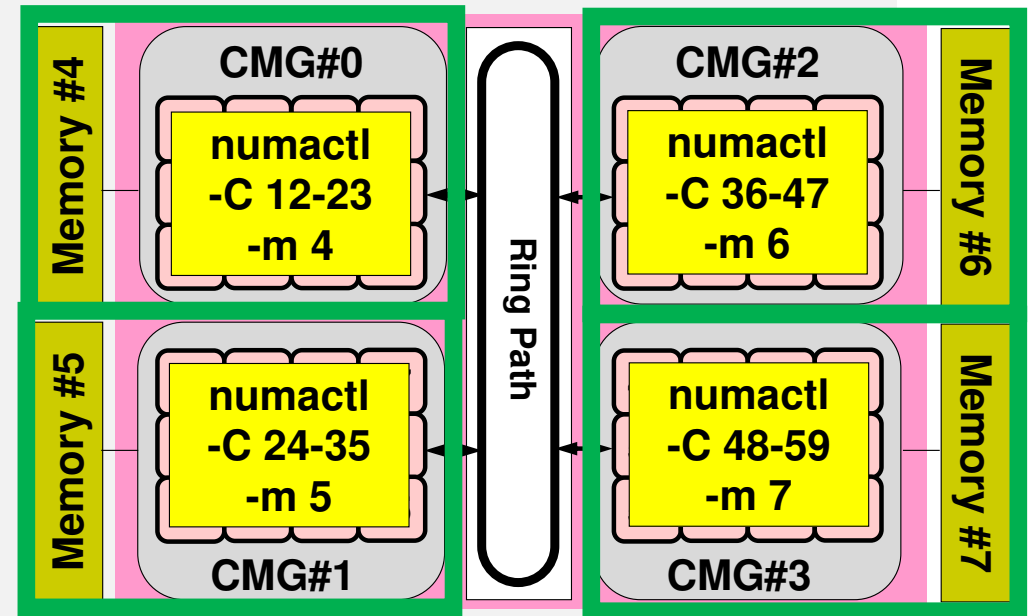
```
numactl -C 12-59 -m 4-7 ./solc2
```

```
numactl -C 12-59 -m 4-7 ./solc2
```

```
numactl -C 12-59 -m 4-7 ./solc2
```

```
numactl -C 12-59 -m 4-7 ./solc2
```

```
numactl -C 12-59 -m 4-7 ./solc2
```



Results (1/2): Fortran

```
[t00XYZ@wisteria01 run]$ cat f12.lst
```

```

  1      8.958216E+00
101      8.313496E+00
201      2.090443E+00
301      3.811029E-01
401      3.769653E-02
501      9.429978E-04
601      4.940783E-05
701      1.888611E-06
801      2.243179E-08
826      9.818026E-09
      5.275418E+00 sec. (solver)

```

```
##ANSWER      2097152      1.459831E+04
```

```

  1      8.958216E+00
101      8.313496E+00
201      2.090443E+00
301      3.811029E-01
401      3.769653E-02
501      9.429978E-04
601      4.940783E-05
701      1.888611E-06
801      2.243179E-08
826      9.818026E-09
      5.270398E+00 sec. (solver)

```

```
##ANSWER      2097152      1.459831E+04
```

Results (1/2): Best for 5 Measurements

```
[XYZ@wisteria01 run]$ grep "(sol" f1_48.lst
```

```
1.480524E+00 sec. (solver)
1.501454E+00 sec. (solver)
1.441297E+00 sec. (solver)
1.483405E+00 sec. (solver)
1.481864E+00 sec. (solver)
```

```
numactl ./solf1
```

```
1.475129E+00 sec. (solver)
1.483695E+00 sec. (solver)
1.485036E+00 sec. (solver)
1.502549E+00 sec. (solver)
1.487192E+00 sec. (solver)
```

```
numactl -C 12-59 -m 4-7 ./solf1
```

```
[XYZ@wisteria01 run]$ grep "(sol" f2_48.lst
```

```
7.713702E-01 sec. (solver)
7.568300E-01 sec. (solver)
7.328739E-01 sec. (solver)
7.826090E-01 sec. (solver)
7.884219E-01 sec. (solver)
```

```
numactl ./solf2
```

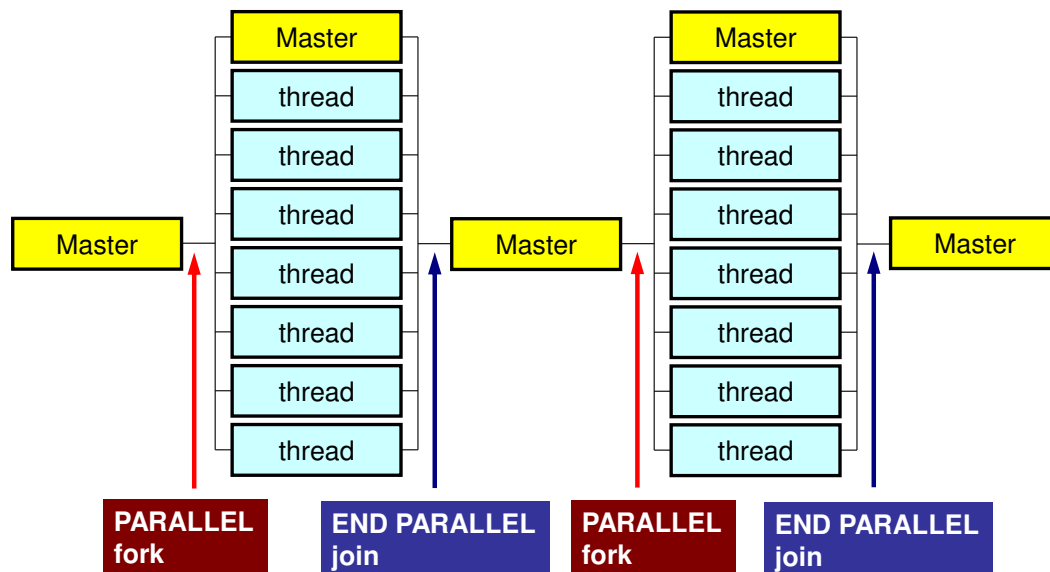
```
7.546160E-01 sec. (solver)
7.937970E-01 sec. (solver)
7.403760E-01 sec. (solver)
7.745121E-01 sec. (solver)
7.862871E-01 sec. (solver)
```

```
numactl -C 12-59 -m 4-7 ./solf2
```

- OpenMP
- Login to Wisteria/BDEC-01
- Parallel Code by OpenMP (0): up to 12 cores
- Parallel Code by OpenMP (1): First Touch
- Parallel Code by OpenMP (2): +ELL
- **Parallel Code by OpenMP (3): reduced omp-parallel**
- **Parallel Code by OpenMP (4): Further Optimization (Fortran only)**

omp parallel (do)

- “omp parallel-omp end parallel” = “fork-join”
- If you have many loops, these “fork-join’s” cause overheads
- **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
```

!\$omp parallel do: Fork-Join

```
#pragma omp parallel for private (i,VAL,j)
for(i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for(j=indexLU[i]; j<indexLU[i+1]; j++) {
        VAL += AMAT[j] * W[P][itemLU[j]];
    }
    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 $i = 1, 2, \dots$

 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

$q^{(i)} = [A]p^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

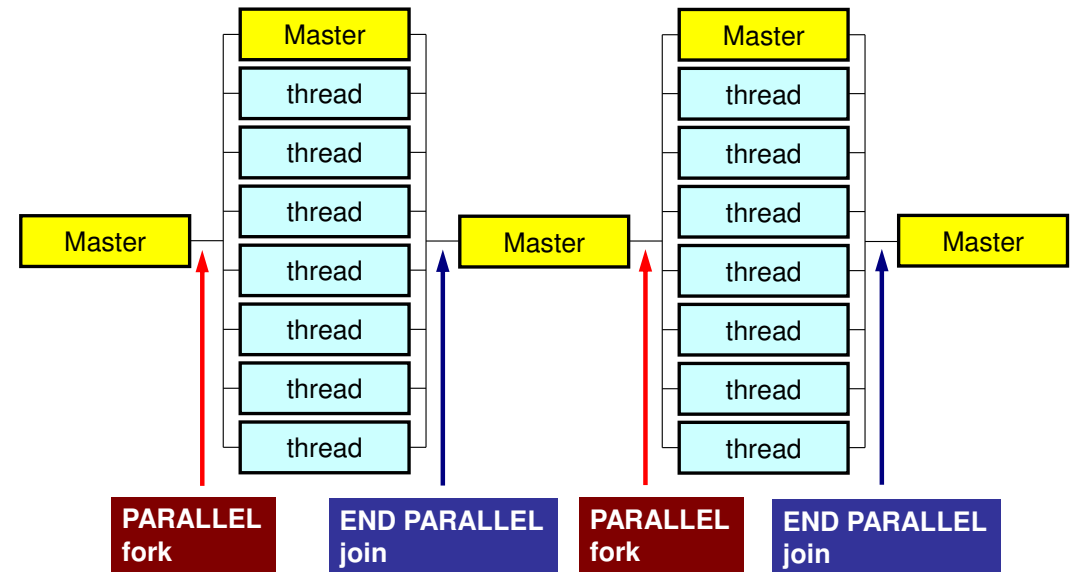
$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

check convergence $|r|$

end

Strategy for Further Optimization

- src-c3, src-f3
 - Only 1 omp-parallel in each iteration
- src-f4 (Fortran Only)
 - Only 1 !\$omp parallel during PCG
 - NO !\$omp do
 - Operations for reduction in dot-product's are NOT parallelized



Further Optimization

src-f3

```
do L= 1, ITR  
!$omp parallel private (i,k,VAL)  
(...)  
!$omp end parallel  
enddo
```

900 continue

ITR= L

deallocate (W)

return
end

src-f4

!\$omp parallel private (...)

```
do L= 1, ITR  
(...)  
enddo
```

900 continue

ITR= L

!\$omp end parallel

deallocate (W)

return
end

src_c3 (1/2)

```

*ITR = N;

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

#pragma omp parallel private (i,j,VAL) {

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

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

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

```

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

for $i = 1, 2, \dots$

$\text{solve } [M]z^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)} \cdot z^{(i-1)}$

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

$q^{(i)} = [A]p^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

check convergence $|r|$

end

src_c3 (2/2)

```

#pragma omp for
for(i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for(i=0; i<6; i++) {
        VAL += AMAT[6*i+j] * W[P][itemLU[6*i+j]];
    }
    W[Q][i] = VAL;
}

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

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

DNRM2 = 0.0;
#pragma omp for 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 $i = 1, 2, \dots$

 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

Program for “src-c3”

```
>$ cd /work/gt80/t80XYZ/ompw  
>$ cd run
```

```
<modify "INPUT.DAT", "c3_48.sh">
```

```
>$ pjsub c3_48.sh
```

c3_48.sh

```
#!/bin/sh
#PJM -N "c3_48"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=48
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o c3_48.1st
```

```
module load fj
```

```
export OMP_NUM_THREADS=48
```

```
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand
```

```
numactl ./solc3
```

```
numactl ./solc3
```

```
numactl ./solc3
```

```
numactl ./solc3
```

```
numactl ./solc3
```

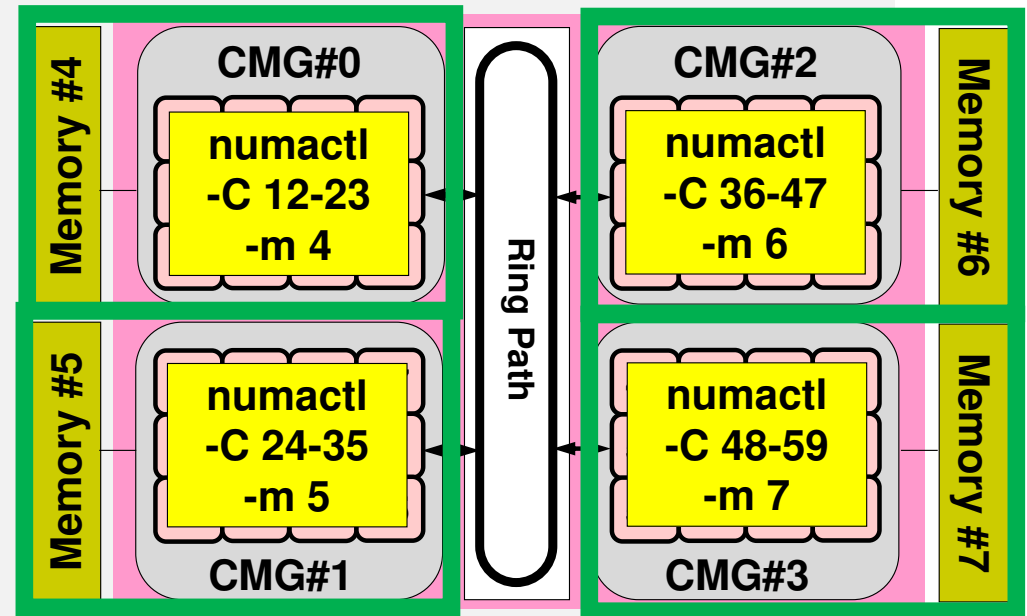
```
numactl -C 12-59 -m 4-7 ./solc3
```

```
numactl -C 12-59 -m 4-7 ./solc3
```

```
numactl -C 12-59 -m 4-7 ./solc3
```

```
numactl -C 12-59 -m 4-7 ./solc3
```

```
numactl -C 12-59 -m 4-7 ./solc3
```



C Language: trad

```
>$ cd /work/gt80/t80XYZ/ompw  
>$ cd run  
  
<modify "INPUT.DAT", "c48org.sh">  
  
>$ pjsub c48org.sh
```

c48org.sh

```
#!/bin/sh  
#PJM -N "cx48"  
#PJM -L rscgrp=lecture0-o  
#PJM -L node=1  
#PJM --omp thread=48  
#PJM -L elapse=00:15:00  
#PJM -g gt80  
#PJM -j  
#PJM -e err  
#PJM -o c48org_160.lst  
  
module load fj  
export OMP_NUM_THREADS=48  
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand  
  
numactl ./solc0org  
numactl ./solc0org  
numactl ./solc0org  
numactl ./solc0org  
numactl ./solc0org  
...
```

Time for PCG, $N=128^3$, 48 Threads

$N= 2,097,152$

	Fortran	C (clang)	C (trad)
src0	1.671	1.564	2.354
src1 (First Touch)	1.480	1.122	1.720
src2 (+ ELL)	0.747	0.809	1.127
src3 (+ reduced “omp-parallel”)	0.707	0.834	0.854

Time for PCG, $N=160^3$, 48 Threads

$N=4,096,000$

	Fortran	C (clang)	C (trad)
src0	3.610	3.484	4.067
src1 (First Touch)	2.993	2.228	3.425
src2 (+ ELL)	1.534	1.690	2.340
src3 (+ reduced “omp-parallel”)	1.556	1.693	1.742

Time for PCG, $N=200^3$, 48 Threads

$N= 8,000,000$

	Fortran	C (clang)	C (trad)
src0	7.666	8.321	9.397
src1 (First Touch)	6.952	5.102	8.008
src2 (+ ELL)	3.421	3.910	5.381
src3 (+ reduced “omp-parallel”)	3.440	3.920	3.824

Time for PCG, $N=256^3$, 48 Threads

$N= 16,777,216$

	Fortran	C (clang)	C (trad)
src0	34.308	24.772	25.547
src1 (First Touch)	32.202	22.172	23.814
src2 (+ ELL)	8.916	10.761	14.566
src3 (+ reduced “omp-parallel”)	8.915	10.764	10.415

src_f4 (1/5)

parallel computing by OpenMP

```

module solver_PCG
  contains
!C
!C*** solve_PCG
!C
    subroutine solve_PCG                                     &
      &      ( N, NPLU, indexLU, itemLU, D, B, X, AMAT, EPS, ITR, IER)

      use omp_lib
      implicit REAL*8 (A-H, O-Z)

      real(kind=8), dimension(N)      :: D
      real(kind=8), dimension(N)      :: B
      real(kind=8), dimension(N)      :: X
      real(kind=8), dimension(NPLU)   :: AMAT

      integer, dimension(0:N)         :: indexLU
      integer, dimension(NPLU)        :: itemLU

      real(kind=8), dimension(:, :), allocatable :: W
      integer(kind=4), dimension(:), allocatable :: SMPindex

      integer, parameter :: R= 1
      integer, parameter :: Z= 2
      integer, parameter :: Q= 2
      integer, parameter :: P= 3
      integer, parameter :: DD= 4

      real(kind=8), dimension(:), allocatable :: W_RH0, W_C1, W_DNRM2

```

src_f4 (2/5)

```

allocate (W(N+N2, 4))

!$omp parallel do private(i)
do i= 1, N
  X(i) = 0. d0
  W(i, 2)= 0. 0D0
  W(i, 3)= 0. 0D0
  W(i, DD)= 1. d0/D(i)
enddo
!$omp parallel do private(i)
do i= N+1, N+N2
  X(i) = 0. d0
  W(i, 2)= 0. 0D0
  W(i, 3)= 0. 0D0
  W(i, DD)= 1. d0/D(i)
enddo

```

```

!$omp parallel
  PEsmptOT= omp_get_num_threads()
!$omp end parallel

```

PEsmptOT: Total Number of Threads

```

allocate (SMPindex(0:PEsmptOT))
SMPindex(0)= 0
m = N/PEsmptOT
nr= N - PEsmptOT*m
do ip= 1, PEsmptOT
  SMPindex(ip)= m
  if (ip.le.nr) SMPindex(ip)= m+1
enddo
do ip= 1, PEsmptOT
  SMPindex(ip)= SMPindex(ip)+SMPindex(ip-1)
enddo

```

SMPindex(0:PEsmptOT): Element# for each thread

```

allocate (W_RH0(PEsmptOT), W_C1(PEsmptOT), W_DNRM2(PEsmptOT))

```

内積用

src_f4 (3/5)

```
!$omp parallel do private(i, VAL, k)  
  do i= 1, N  
    VAL= D(i)*X(i)  
    do k= 1, 6  
      VAL= VAL + AMAT(k, i)*X(itemLU(k, i))  
    enddo  
    W(i, R)= B(i) - VAL  
  enddo  
  
  BNRM2= 0.0D0  
!$omp parallel do private(i) reduction(+:BNRM2)  
  do i= 1, N  
    BNRM2 = BNRM2 + B(i) **2  
  enddo
```

```

ITR= N
Stime= omp_get_wtime()

!$omp parallel private(L, ip, ip1, ip2, i, k, VAL)
!$omp& private(RHO, BETA, RH01, C1, ALPHA, DNRM2)
do L= 1, ITR

    ip = omp_get_thread_num()+1
    ip1= SMPindex(ip-1)+1
    ip2= SMPindex(ip)

!$omp simd
do i= ip1, ip2
    W(i, Z)= W(i, R)*W(i, DD)
enddo

    W_RHO(ip)= 0.0d0
!$omp simd
do i= ip1, ip2
    W_RHO(ip)= W_RHO(ip) + W(i, R)*W(i, Z)
enddo
!$omp barrier
    RHO= 0.d0
!$omp simd
do i= 1, PEsmptTOT
    RHO= RHO + W_RHO(i)
enddo

    if ( L.eq.1 ) then
!$omp simd
do i= ip1, ip2
    W(i, P)= W(i, Z)
enddo
    else
        BETA= RHO / RH01
!$omp simd
do i= ip1, ip2
    W(i, P)= W(i, Z) + BETA*W(i, P)
enddo
    endif
!$omp barrier

```

src_f4 (4/5)

```

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

```

```

do i= ip1, ip2
  VAL= D(i)*W(i,P)
  do k= 1, 6
    VAL= VAL + AMAT(k,i)*W(itemLU(k,i),P)
  enddo
  W(i,Q)= VAL
enddo

W_C1(ip)= 0.0d0
!$omp simd
do i= ip1, ip2
  W_C1(ip)= W_C1(ip) + W(i,P)*W(i,Q)
enddo
!$omp barrier
C1= 0.d0
!$omp simd
do i = 1, PEsmptOT
  C1= C1 + W_C1(i)
enddo
ALPHA= RH0 / C1

!$omp simd
do i= ip1, ip2
  X(i) = X(i) + ALPHA * W(i,P)
  W(i,R)= W(i,R) - ALPHA * W(i,Q)
enddo

W_DNRM2(ip)= 0.0d0
!$omp simd
do i= ip1, ip2
  W_DNRM2(ip)= W_DNRM2(ip) + W(i,R)**2
enddo
!$omp barrier
DNRM2= 0.d0
!$omp simd
do i = 1, PEsmptOT
  DNRM2= DNRM2 + W_DNRM2(i)
enddo
ERR = dsqrt(DNRM2/BNRM2)...

```

src_f4 (5/5)

```

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

```

Program for “src4”

```
>$ cd /work/gt80/t80XYZ/ompw
```

```
>$ cd run
```

```
<modify "INPUT.DAT", "f4_48.sh">
```

```
>$ pjsub f4_48.sh
```


f4_48.sh

```
#!/bin/sh
#PJM -N "f4_48"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=48
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o f4_48.1st
```

```
module load fj
```

```
export OMP_NUM_THREADS=48
```

```
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand
```

```
numactl ./solf4
```

```
numactl ./solf4
```

```
numactl ./solf4
```

```
numactl ./solf4
```

```
numactl ./solf4
```

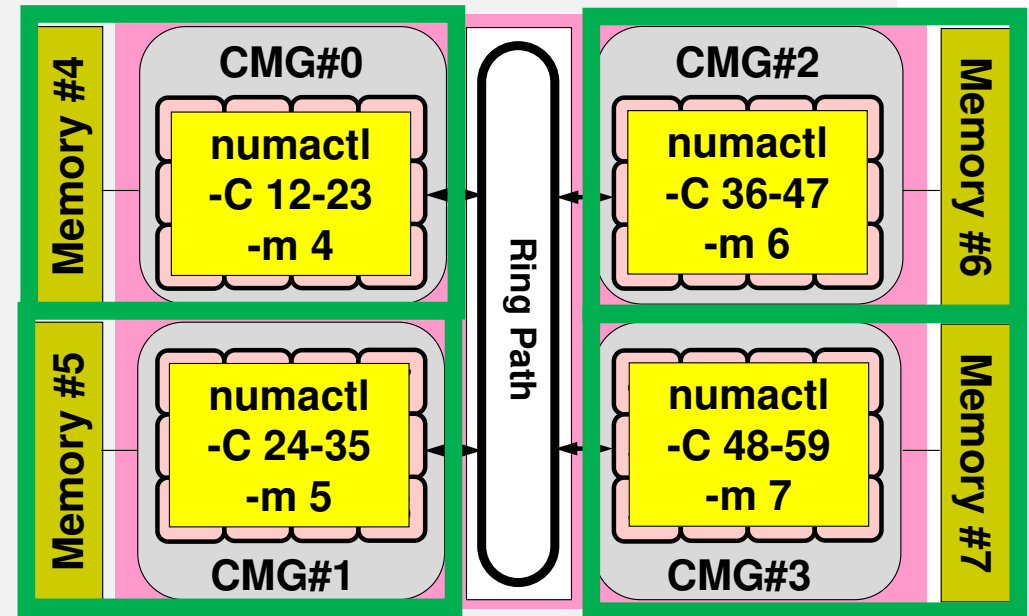
```
numactl -C 12-59 -m 4-7 ./solf4
```

```
numactl -C 12-59 -m 4-7 ./solf4
```

```
numactl -C 12-59 -m 4-7 ./solf4
```

```
numactl -C 12-59 -m 4-7 ./solf4
```

```
numactl -C 12-59 -m 4-7 ./solf4
```



Time for PCG: $N=128^3$, 48 Threads

$N=2,097,152$

	Fortran	C (clang)	C (trad)
src0	1.671	1.564	2.354
src1 (First Touch)	1.480	1.122	1.720
src2 (+ ELL)	0.747	0.809	1.127
src3 (+ reduced "omp-parallel")	0.707	0.834	0.854
src3b (+ clang loop unroll_count)	-	0.764	-
src4 (+ No OMP-DO/Reduction)	0.676	-	-

src_c3b (5/5)

src-c3

```
#pragma omp for
for(i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for(j=0; j<6; j++) {
        VAL += AMAT[6*i+j]*W[P][itemLU[6*i+j]];
    }
    W[Q][i] = VAL;
}
```

src-c3b: clang only

```
#pragma omp for
#pragma clang loop unroll_count(8)
for(i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for(j=0; j<6; j++) {
        VAL += AMAT[6*i+j]*W[P][itemLU[6*i+j]];
    }
    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

Time for PCG: $N=160^3$, 48 Threads

$N= 4,096,000$

	Fortran	C (clang)	C (trad)
src0	3.610	3.484	4.067
src1 (First Touch)	2.993	2.228	3.425
src2 (+ ELL)	1.534	1.690	2.340
src3 (+ reduced “omp-parallel”)	1.556	1.693	1.742
src3b (+ clang loop unroll_count)	-	1.586	-
src4 (+ No OMP-DO/Reduction)	1.435	-	-

Time for PCG: $N=200^3$, 48 Threads

$N=8,000,000$

	Fortran	C (clang)	C (trad)
src0	7.666	8.321	9.397
src1 (First Touch)	6.952	5.102	8.008
src2 (+ ELL)	3.421	3.910	5.381
src3 (+ reduced "omp-parallel")	3.440	3.920	3.824
src3b (+ clang loop unroll_count)	-	3.624	-
src4 (+ No OMP-DO/Reduction)	3.276	-	-

Time for PCG: $N=256^3$, 48 Threads

$N=16,777,216$

	Fortran	C (clang)	C (trad)
src0	34.308	24.772	25.547
src1 (First Touch)	32.202	22.172	23.814
src2 (+ ELL)	8.916	10.761	14.566
src3 (+ reduced "omp-parallel")	8.915	10.764	10.415
src3b (+ clang loop unroll_count)	-	10.003	-
src4 (+ No OMP-DO/Reduction)	8.620	-	-

Exercises

- Problem size (NX, NY, NZ)
- Thread # (OMP_NUM_THREADS: 1-48)
- Various Types of Implementation
- Profiling

Performance Evaluation: Profiler

- Specify the measuring unit in the program (multiple settings are possible)
- 17 runs needed for each case
- Performance of Computation & Memory, Power Consumption
- Excel Macro File
 - https://www.dropbox.com/s/kat9ny5aoxp7cqm/cpu_pa_report.xlsm?dl=0

Profiling (1/4)

Specifying the measuring unit in the program, No additional options for compiling: solver_PCG.c/f

```
#include "fj_tool/fapp.h"

fapp_start ("CG",1,0);
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();
fapp_stop ("CG",1,0);

return 0;
}
```



```
call fapp_start ("CG",1,0)
Stime = omp_get_wtime()

do L= 1, ITR
...
    if (ERR .lt. EPS) then
        IER = 0
        goto 900
    else
        RH01 = RH0
    endif

enddo
IER = 1
900 continue
Etime= omp_get_wtime()
call fapp_stop ("CG",1,0)

return
end
```



Profiling (2/4)

Running on Wisteria/BDEC-01 (Odyssey)

```
>$ cd /work/gt80/t80XYZ/ompw  
>$ cd run
```

```
<modify "fapp.sh", "data.sh">
```

```
>$ pjsub fapp.sh
```

(after finishing)

```
>$ pjsbu data.sh
```

```
>$ ls pa*.csv  
pa1.csv ... pa17.csv
```

Profiling (3/4)

Each Directory (e.g. repo01 etc.) should be empty before running the code

fapp.sh

```
#!/bin/sh
#PJM -N "fapp"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM -omp thread=48
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
export OMP_NUM_THREADS=48
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand
```

```
fapp -C -d ./repo01 -Hevent=pa1 ./solf2
fapp -C -d ./repo02 -Hevent=pa2 ./solf2
fapp -C -d ./repo03 -Hevent=pa3 ./solf2
fapp -C -d ./repo04 -Hevent=pa4 ./solf2
fapp -C -d ./repo05 -Hevent=pa5 ./solf2
fapp -C -d ./repo06 -Hevent=pa6 ./solf2
fapp -C -d ./repo07 -Hevent=pa7 ./solf2
fapp -C -d ./repo08 -Hevent=pa8 ./solf2
fapp -C -d ./repo09 -Hevent=pa9 ./solf2
fapp -C -d ./repo10 -Hevent=pa10 ./solf2
fapp -C -d ./repo11 -Hevent=pa11 ./solf2
fapp -C -d ./repo12 -Hevent=pa12 ./solf2
fapp -C -d ./repo13 -Hevent=pa13 ./solf2
fapp -C -d ./repo14 -Hevent=pa14 ./solf2
fapp -C -d ./repo15 -Hevent=pa15 ./solf2
fapp -C -d ./repo16 -Hevent=pa16 ./solf2
fapp -C -d ./repo17 -Hevent=pa17 ./solf2
```

data.sh

```
#!/bin/sh
#PJM -N "data"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --mpi proc=1
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o data.lst
```

```
module load fj
module load fjmp
```

```
fapp -A -d ./repo01 -Icpupa,mpi -tcsv -o pa1.csv
fapp -A -d ./repo02 -Icpupa,mpi -tcsv -o pa2.csv
fapp -A -d ./repo03 -Icpupa,mpi -tcsv -o pa3.csv
fapp -A -d ./repo04 -Icpupa,mpi -tcsv -o pa4.csv
fapp -A -d ./repo05 -Icpupa,mpi -tcsv -o pa5.csv
fapp -A -d ./repo06 -Icpupa,mpi -tcsv -o pa6.csv
fapp -A -d ./repo07 -Icpupa,mpi -tcsv -o pa7.csv
fapp -A -d ./repo08 -Icpupa,mpi -tcsv -o pa8.csv
fapp -A -d ./repo09 -Icpupa,mpi -tcsv -o pa9.csv
fapp -A -d ./repo10 -Icpupa,mpi -tcsv -o pa10.csv
fapp -A -d ./repo11 -Icpupa,mpi -tcsv -o pa11.csv
fapp -A -d ./repo12 -Icpupa,mpi -tcsv -o pa12.csv
fapp -A -d ./repo13 -Icpupa,mpi -tcsv -o pa13.csv
fapp -A -d ./repo14 -Icpupa,mpi -tcsv -o pa14.csv
fapp -A -d ./repo15 -Icpupa,mpi -tcsv -o pa15.csv
fapp -A -d ./repo16 -Icpupa,mpi -tcsv -o pa16.csv
fapp -A -d ./repo17 -Icpupa,mpi -tcsv -o pa17.csv
```

Profiling (4/4): Operations on PC

- Copying all “pa*.csv”s to your PC

```
>$ scp t00XYZ@wisteria.cc.u-tokyo.ac.jp:/work/gt00/t00XYZ/ompw/run/pa*.csv .
```

- All “pa*.csv”s and Macro for Excel should be in the same directory
- “Double Click” the Excel Macro
 - Just follow instructions
 - Please select “CG”, not “All”

Time for PCG: $N=160^3$, 48 Threads

$N= 4,096,000$, for Each Node, Fortran
Optimization – Busy Memory – More Watt

	Time (sec.)	Peak Perf. Ratio (%)	SIMD Ratio (%)	Memory Throughput (%)	Instruction		Power (W)	
					Effective	Load/ Store	Core L2 Memory	Node
solf0	3.69	1.59	20.0	30.3	3.39×10^{11}	8.27×10^{10}	81.6 10.9 20.2	112.
solf1 (First Touch)	2.98	1.97	28.8	37.5	2.35×10^{11}	5.33×10^{10}	92.1 10.8 33.3	136.
solf2 (+ ELL)	1.58	3.73	49.7	70.0	1.19×10^{11}	4.17×10^{10}	104. 15.0 51.7	170.
solf3 (+ reduced “omp- parallel”)	1.58	3.72	48.4	69.8	1.22×10^{11}	4.10×10^{10}	101. 14.9 51.0	167.
solf4 (+ further optimization)	1.45	4.06	55.1	75.8	1.10×10^{11}	3.78×10^{10}	102. 15.8 56.5	174.

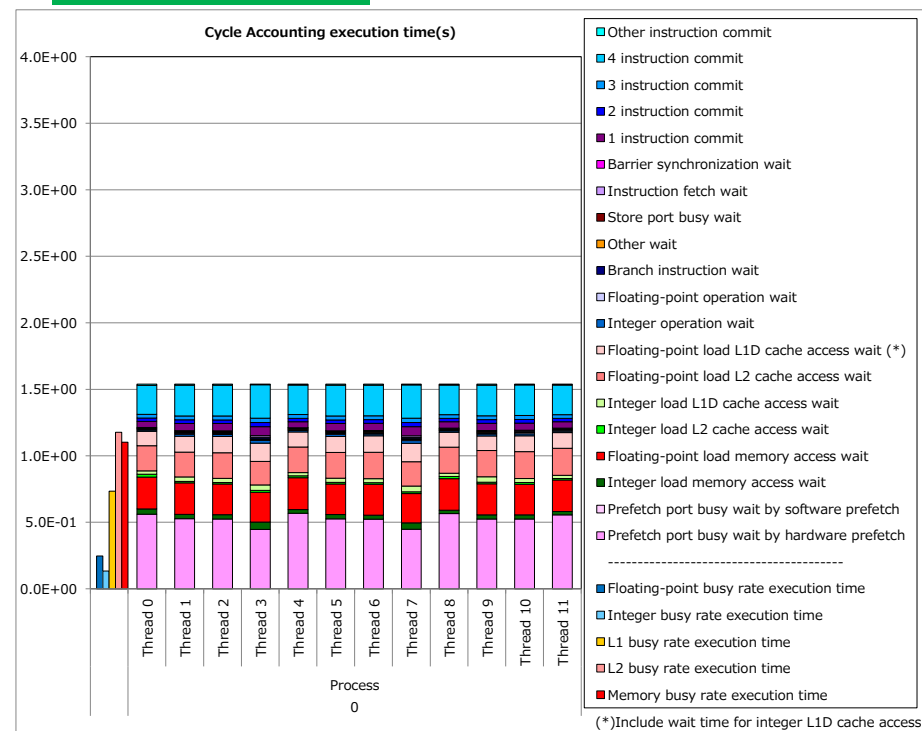
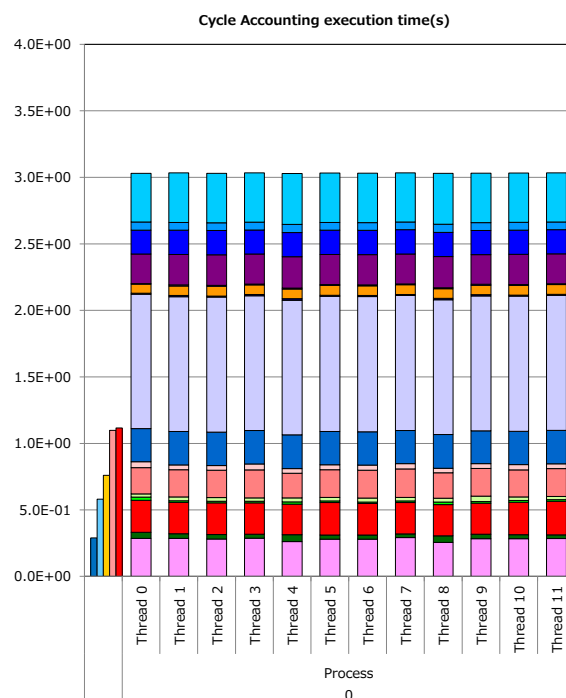
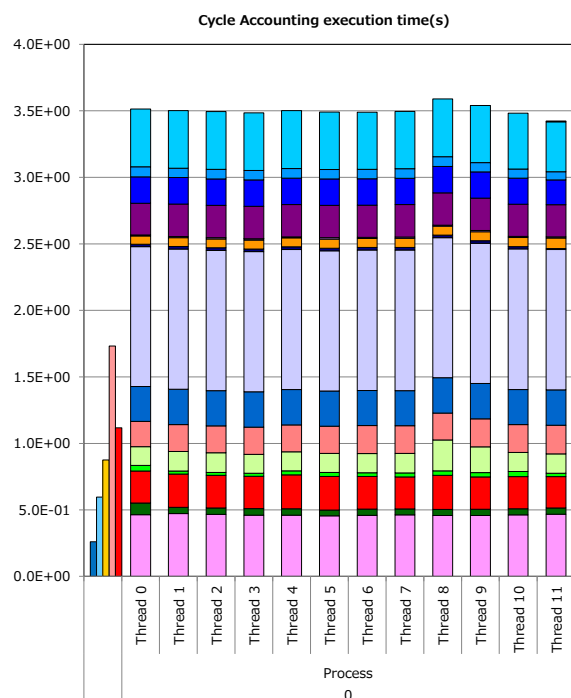
PCG, $N=160^3$, 48 Threads

Fortran, Each CMG

src0:Initial

src1:First Touch

src2: +ELL



■ Prefetch Port Busy Wait (H/W)
■ Integer Load L1D Cache Access Wait
■ Floating-Point Operation Wait
■ Instruction Commit

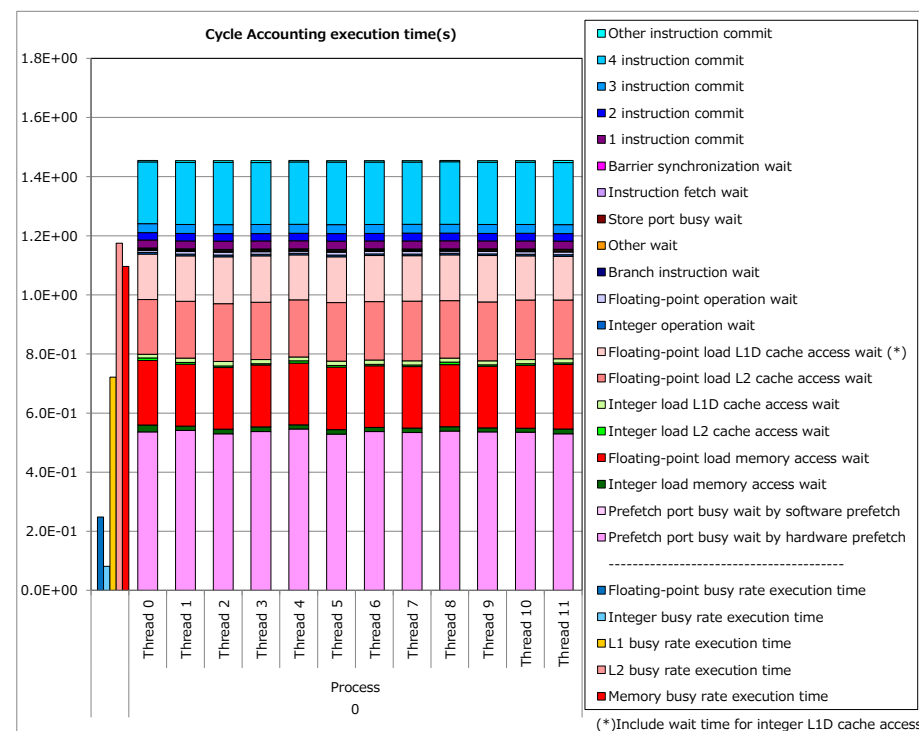
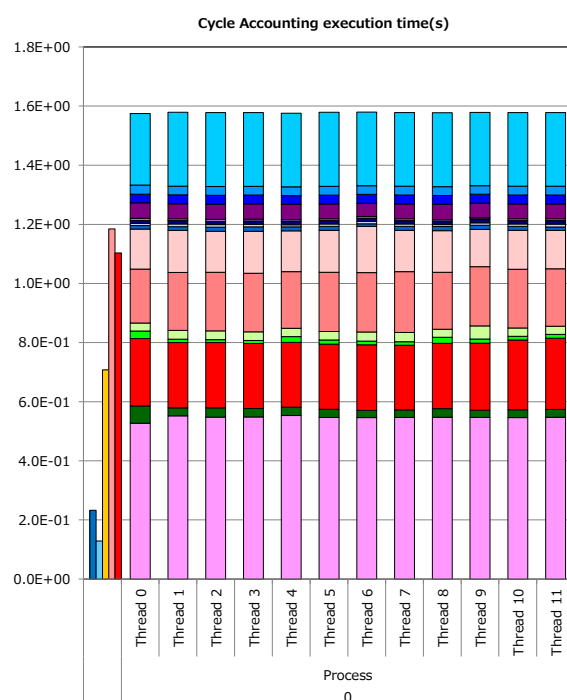
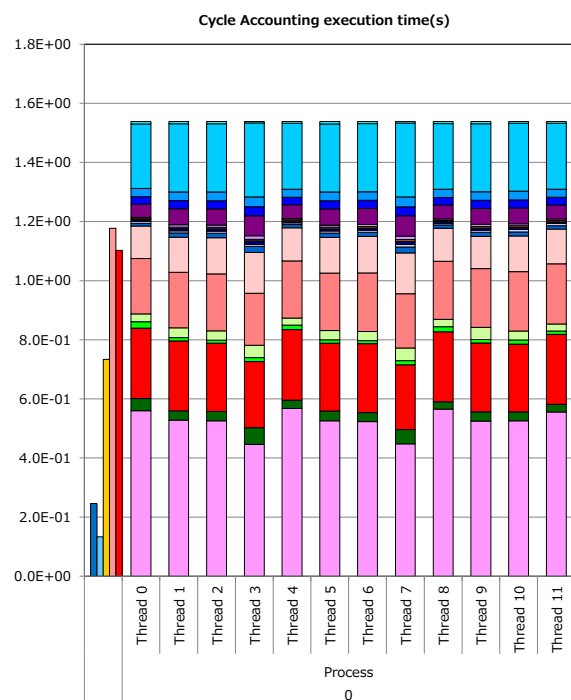
PCG, $N=160^3$, 48 Threads

Fortran, Each CMG

src2:+ELL

src3

src4



Prefetch Port Busy Wait (H/W)
 Floating-Point Load Memory Access Wait
 Floating-Point Load L2D Cache Access Wait
 Floating-Point Load L1D Cache Access Wait
 Instruction Commit

Time for PCG: $N=160^3$, 48 Threads

$N= 4,096,000$, for Each Node, C (clang)

Optimization – Busy Memory – More Watt

	Time (sec.)	Peak Perf. Ratio (%)	SIMD Ratio (%)	Memory Throughput (%)	Instruction		Power (W)	
					Effective	Load/Store	Core L2 Memory	Node
solc0	3.53	0.90	2.95	31.7	4.83×10^{11}	1.65×10^{11}	103. 12.9 17.9	134.
solc1 (First Touch)	2.42	1.34	4.82	46.2	2.97×10^{11}	1.10×10^{11}	107. 11.9 33.8	153.
solc2 (+ ELL)	1.73	1.88	8.21	63.8	1.74×10^{11}	8.92×10^{10}	104. 13.9 44.8	163.
solc3 (+ reduced “omp-parallel”)	1.75	1.86	8.51	62.9	1.68×10^{11}	8.88×10^{10}	108. 14.2 45.0	167.
solc3b (+ clang loop unroll_count)	1.62	2.00	10.4	67.8	1.37×10^{11}	8.22×10^{10}	109. 14.6 44.4	168.

Time for PCG: $N=160^3$, 48 Threads

$N= 4,096,000$, for Each Node, C (trad)

Optimization – Busy Memory – More Watt

	Time (sec.)	Peak Perf. Ratio (%)	SIMD Ratio (%)	Memory Throughput (%)	Instruction		Power (W)	
					Effective	Load/ Store	Core L2 Memory	Node
solc0org	4.14	2.66	27.3	27.1	3.78×10^{11}	7.40×10^{10}	91.3 11.3 18.4	121.
solc1org (First Touch)	3.52	3.13	37.4	31.8	2.75×10^{11}	6.14×10^{10}	90. 9.86 28.4	128.
solc2org (+ ELL)	2.34	4.70	58.1	47.1	1.63×10^{11}	4.77×10^{10}	95.1 11.7 37.6	144.
solc3org (+ reduced “omp- parallel”)	1.70	3.46	47.1	64.8	1.17×10^{11}	4.14×10^{10}	96.7 13.9 48.7	159.