Introduction to Parallel Programming for Multicore/Manycore Clusters

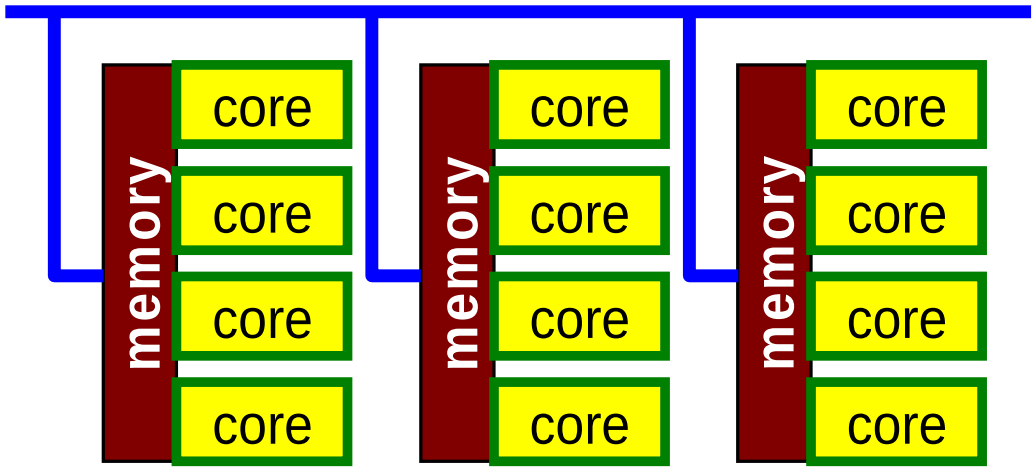
Part II: Introduction to OpenMP

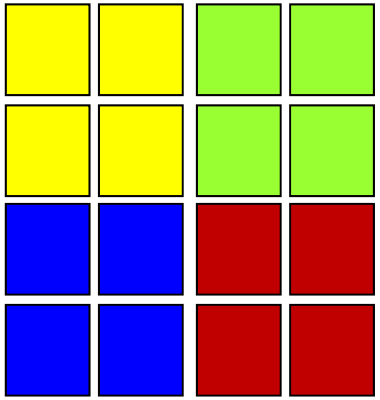
Kengo Nakajima
Information Technology Center
The University of Tokyo

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

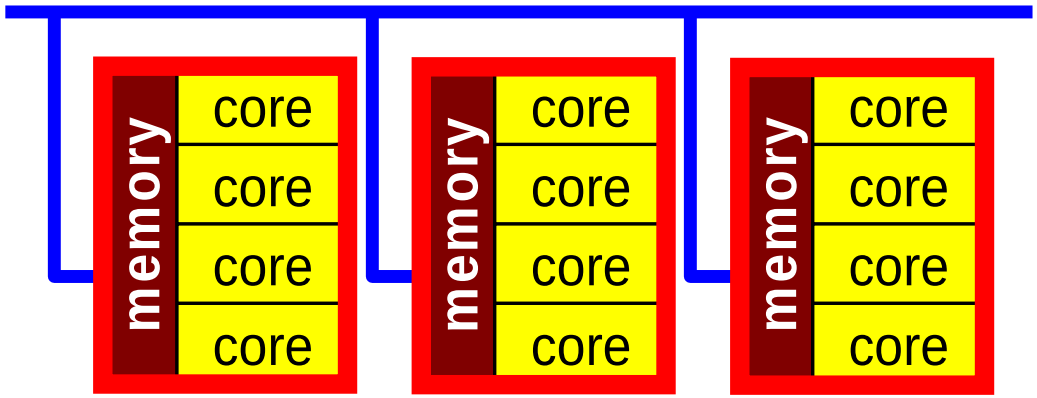
Flat MPI vs. Hybrid

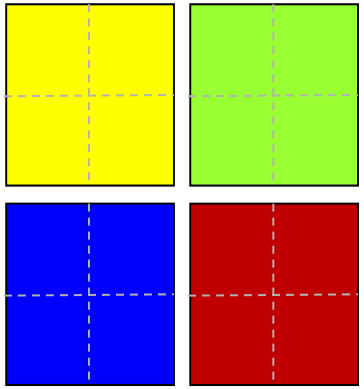
Flat-MPI：Each Core -> Independent

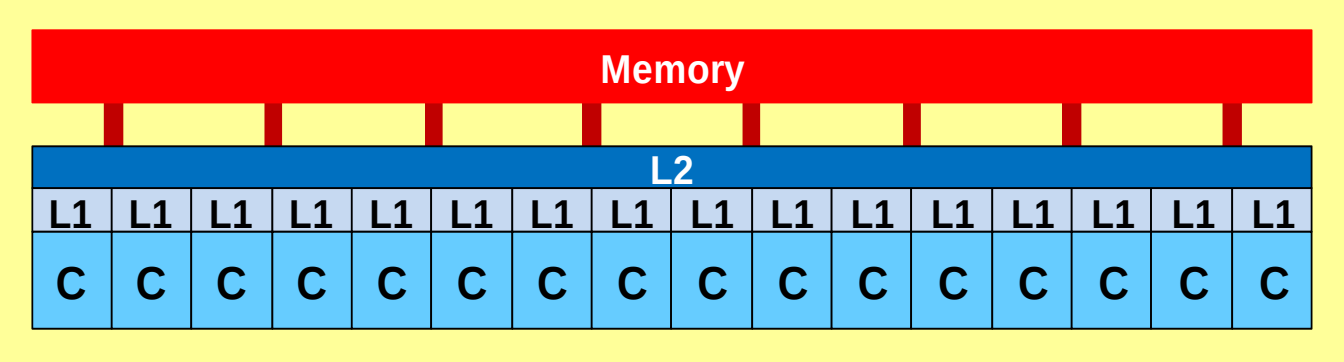




Hybrid：Hierarchal Structure







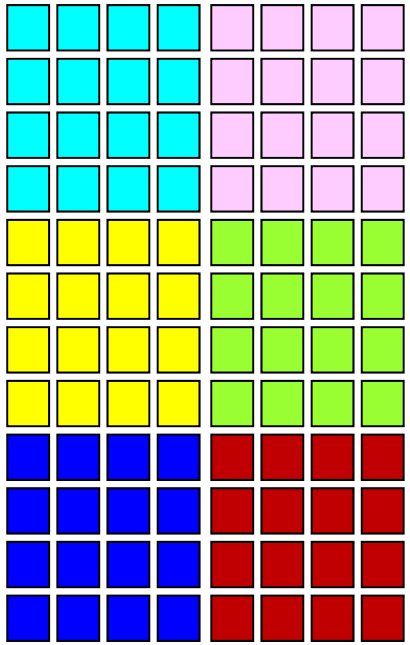
HB M x N

Number of OpenMP threads per a single MPI process

Number of MPI process per a single node

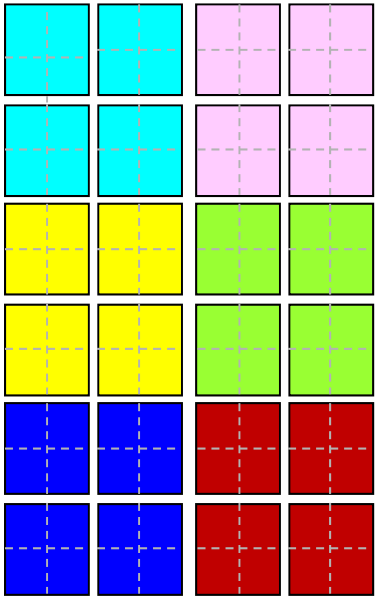
Size of data for each MPI process varies according to HB MxN

example: 6 nodes, 96 cores



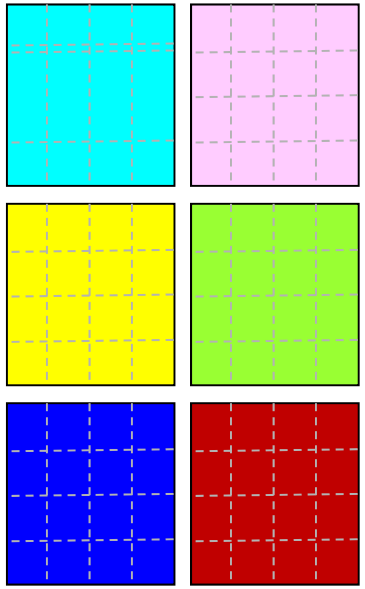
Flat MPI

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



HB 4x4

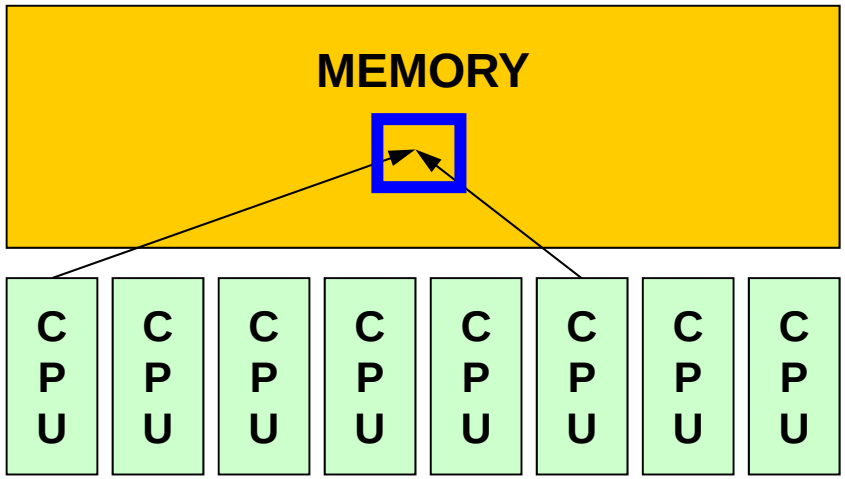
```
128 192 64
  4   6  1
pcube
```



HB 16x1

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

SMP



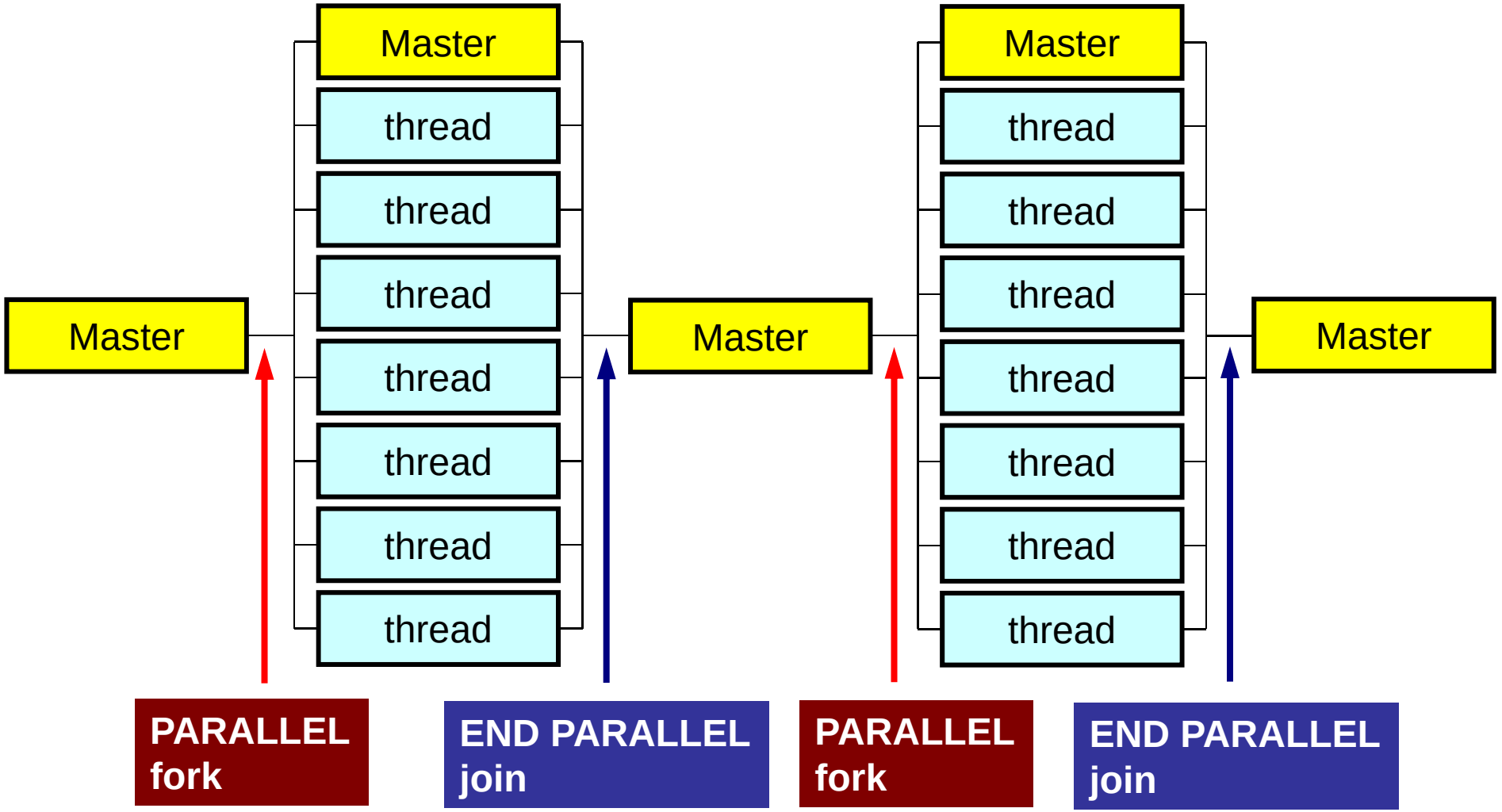
- SMP
  - Symmetric Multi Processors
  - Multiple CPU’s (cores) share a single memory space

What is OpenMP ?

http://www.openmp.org

- An API for multi-platform shared-memory parallel programming in C/C++ and Fortran
  - Current version: 4.X: GPU, Accelerators: close to OpenACC
- Background
  - Merger of Cray and SGI in 1996
  - ASCI project (DOE) 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.
      - SMP Clusters: Intel ASCI Red, IBM Power (Blue, White, Purple)/Blue Gene, SGI
- C/C++ version and Fortran version have been separately developed until ver.2.5.
- Fork-Join Parallel Execution Model
- Users have to specify everything by directives.

Fork-Join Parallel Execution Model

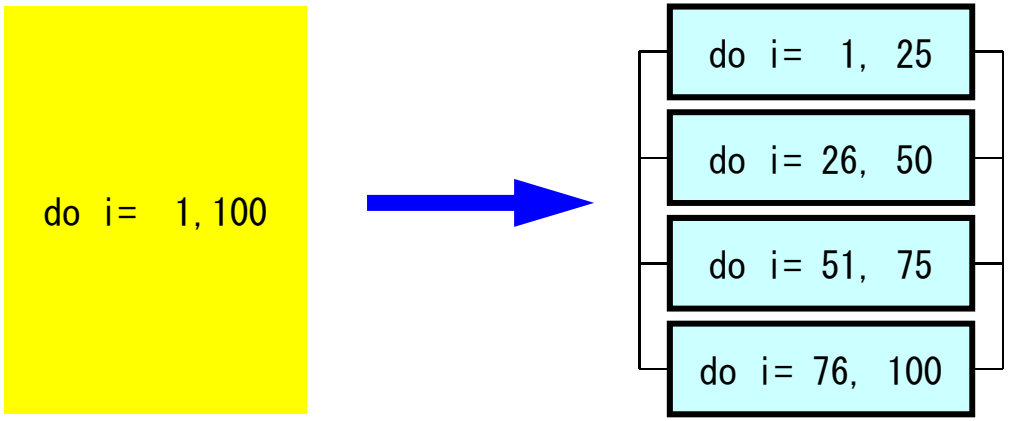


Number of Threads

- **OMP_NUM_THREADS**
  - How to change ?
    - bash(.bashrc)
    - csh(.cshrc)

```
export OMP_NUM_THREADS=8
setenv OMP_NUM_THREADS 8
```

- **OMP_NUM_THREADS=4**



Information about OpenMP

- OpenMP Architecture Review Board (ARB)
  - http://www.openmp.org
- References
  - Chandra, R. et al.「Parallel Programming in OpenMP」（Morgan Kaufmann）
  - Quinn, M.J.「Parallel Programming in C with MPI and OpenMP」（McGrawHill）
  - Mattson, T.G. et al.「Patterns for Parallel Programming」（Addison Wesley）
  - 牛島「OpenMPによる並列プログラミングと数値計算法」（丸善）
  - Chapman, B. et al.「Using OpenMP」（MIT Press）
- 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

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

Dot Products

```
!$omp parallel do private(i)
!$omp&             reduction(+:RHO)
      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo
!$omp end parallel do
```

DAXPY

```
!$omp parallel do
      do i= 1, N
        Y(i)= ALPHA*X(i) + Y(i)
      enddo
!$omp end parallel do
```

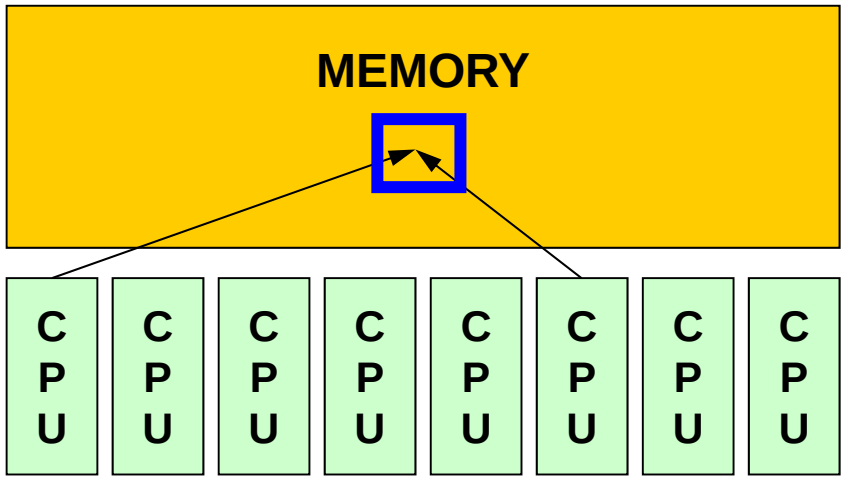
OpenMP/Direceives Matrix/Vector Products

```
!$omp parallel do private(i,j)
      do i= 1, N
        W(i,Q)= D(i)*W(i,P)
        do j= 1, INL(i)
          W(i,Q)= W(i,Q) + W(IAL(j,i),P)
        enddo
        do j= 1, INU(i)
          W(i,Q)= W(i,Q) + W(IAU(j,i),P)
        enddo
      enddo
!$omp end parallel do
```

Features of OpenMP

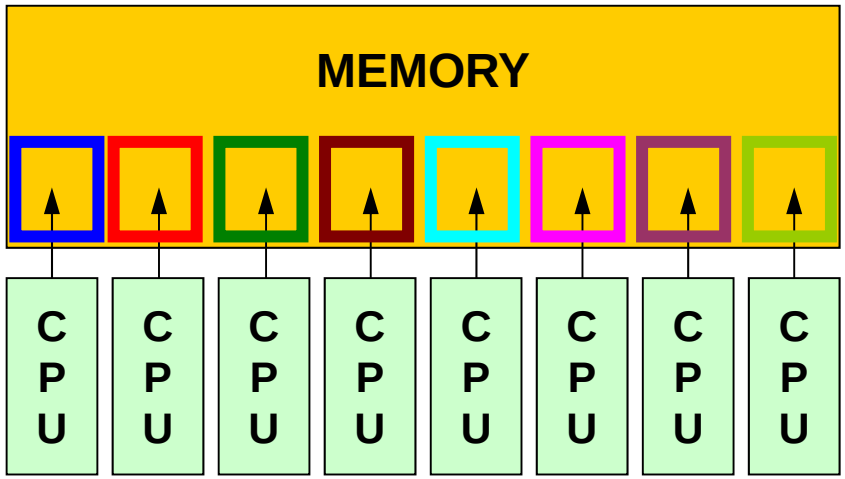
- Directives
  - Loops right after the directives are parallelized.
  - If the compiler does not support OpenMP, directives are considered as just comments.
- Nothing happen without explicit directives
  - Different from “automatic parallelization/vectorization”
  - Something wrong may happen by un-proper way of usage
  - Data configuration, ordering etc. are done under users’ responsibility
- “Threads” are created according to the number of cores on the node
  - Thread: “Process” in MPI
  - Generally, “# threads = # cores”: Xeon Phi supports 4 threads per core (Hyper Multithreading)

Memory Contention: メモリ競合



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

Memory Contention (cont.)



- In this lecture, no such case does not happen by reordering etc.
  - In OpenMP, users are responsible for such issues (e.g. proper data configuration, reordering etc.)
- Generally speaking, performance per core reduces as number of used cores (thread number) increases.
  - Memory access performance: STREAM

Features of OpenMP (cont.)

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

```
!$omp parallel do private(i)
!$omp&             reduction(+:RHO)
      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo
!$omp end parallel do
```

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

FORTRAN & C

```
use omp_lib
...
!$omp parallel do shared(n,x,y) private(i)
      do i= 1, n
        x(i)= x(i) + y(i)
      enddo
!$ omp end parallel do
```

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

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

In this class ...

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

First things to be done

- use omp_lib Fortran
- #include <omp.h> C

OpenMP Directives (Fortran)

```
sentinel directive_name [clause[[,] clause]…]
```

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

```
!$OMP PARALLEL DO
!$OMP+SHARED (A,B,C)
```

```
!$OMP PARALLEL DO &
!$OMP SHARED (A,B,C)
```

OpenMP Directives (C)

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

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

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

PARALLEL DO

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

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

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

REDUCTION

```
REDUCTION ({operator|instinsic}: list)
```

```
reduction ({operator|instinsic}: list)
```

- Similar to “MPI_Reduce”
- Operator
  - +, *, -, .AND., .OR., .EQV., .NEQV.
- Intrinsic
  - MAX, MIN, IAND, IOR, IEQR

Example-1: A Simple Loop

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

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

Example-1: REDUCTION

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

- “END PARALLEL DO” is not required

Functions in OpenMP

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

OpenMP for Dot Products

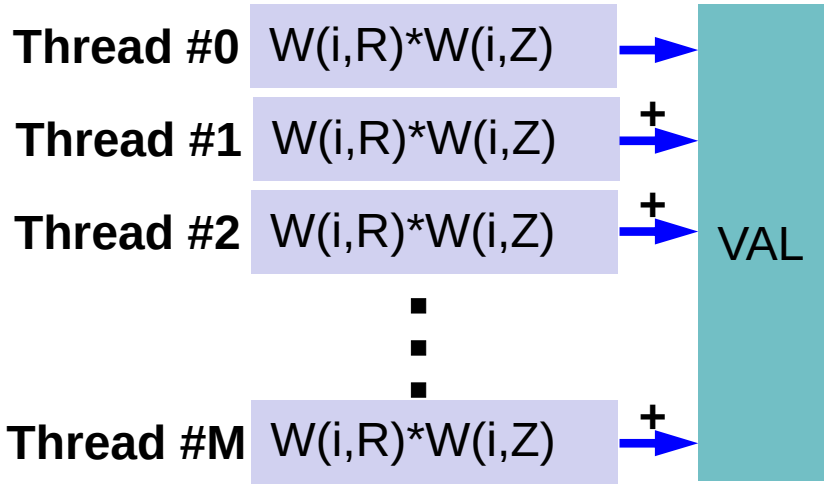
```
VAL= 0.d0
do i= 1, N
  VAL= VAL + W(i,R) * W(i,Z)
enddo
```

OpenMP for Dot Products

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

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

Directives are just inserted.



OpenMP for Dot Products

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

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

Directives are just inserted.

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(ip,i) REDUCTION(+:VAL)
      do ip= 1, PEsmpTOT
        do i= index(ip-1)+1, index(ip)
          VAL= VAL + W(i,R) * W(i,Z)
        enddo
      enddo
!$OMP END PARALLEL DO
```

Multiple Loop
`PEsmpTOT`: Number of threads

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

OpenMP for Dot Products

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(ip,i) REDUCTION(+:VAL)
      do ip= 1, PEsmpTOT
        do i= index(ip-1)+1, index(ip)
          VAL= VAL + W(i,R) * W(i,Z)
        enddo
      enddo
!$OMP END PARALLEL DO
```

Multiple Loop
PEsmpTOT: Number of threads

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

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

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

Matrix-Vector Multiply

```
do i = 1, N
  VAL= D(i)*W(i,P)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL + AL(k)*W(itemL(k),P)
  enddo
  do k= indexU(i-1)+1, indexU(i)
    VAL= VAL + AU(k)*W(itemU(k),P)
  enddo
  W(i,Q)= VAL
enddo
```

Matrix-Vector Multiply

```
!$omp parallel do private(ip,i,VAL,k)
      do ip= 1, PEsmpTOT
        do i = INDEX(ip-1)+1, INDEX(ip)
          VAL= D(i)*W(i,P)
          do k= indexL(i-1)+1, indexL(i)
            VAL= VAL + AL(k)*W(itemL(k),P)
          enddo
          do k= indexU(i-1)+1, indexU(i)
            VAL= VAL + AU(k)*W(itemU(k),P)
          enddo
          W(i,Q)= VAL
        enddo
      enddo
!$omp end parallel do
```

Matrix-Vector Multiply: Other Approach

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

```
!$omp parallel do private(i,VAL,k)
  do i = 1, N
    VAL= D(i)*W(i,P)
    do k= indexL(i-1)+1, indexL(i)
      VAL= VAL + AL(k)*W(itemL(k),P)
    enddo
    do k= indexU(i-1)+1, indexU(i)
      VAL= VAL + AU(k)*W(itemU(k),P)
    enddo
    W(i,Q)= VAL
  enddo
!$omp end parallel do
```

omp parallel (do)

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

```
!$omp parallel ...

!$omp do
   do i= 1, N
...
!$omp do
   do i= 1, N
...
!$omp end parallel     必須
```

```
#pragma omp parallel ...

#pragma omp for {

...
#pragma omp for {
```

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

- OpenMP
- Login to Reedbush-U
- **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, …
    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) + αip(i)
    r(i)= r(i-1) – αiq(i)
    check convergence |r|
end
```

Solving the following equation:

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

“Approximate Inverse Matrix”

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

Ultimate Preconditioning:
Inverse Matrix

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

Diagonal Scaling: Simple but weak

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

Diagonal Scaling, Point-Jacobi

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

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

Files on Reedbush-U

```
>$ cd /lustre/gt27/t27XXX

>$ cp /lustre/gt00/z30088/omp/omp-c.tar . or
>$ cp /lustre/gt00/z30088/omp/omp-f.tar .

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

>$ cd multicore

>$ ls
   omp  stream

Hereafter: <$O-omp>, <$O-stream>
>$ cd omp/src20
>$ make
>$ cd ../run
(modify go1.sh)
>$ qsub go1.sh
```

<$O-omp>/src20/Makefile

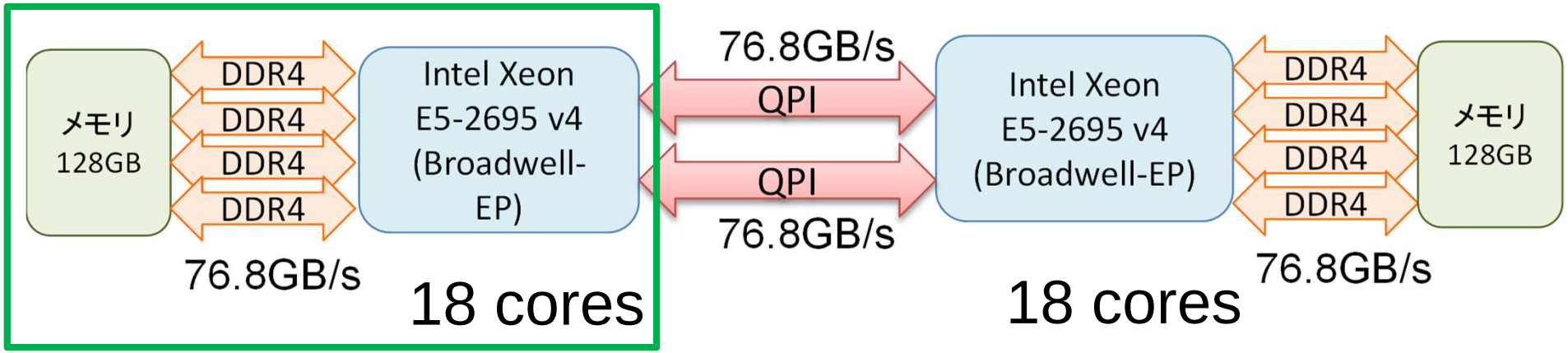
parallel computing by OpenMP

```
F90   = ifort
F90OPTFLAGS= -O3 -qopenmp -ipo -xCORE-AVX2 -align array32byte

F90FLAGS =$(F90OPTFLAGS)

.SUFFIXES:
.SUFFIXES: .o .f .f90 .c
#
.f90.o:; $(F90) -c $(F90FLAGS)  $(F90OPTFLAG) $<
.f.o:; $(F90) -c $(F90FLAGS)  $(F90OPTFLAG) $<
#
OBJS = ¥
solver_PCG.o rcm.o struct.o pcg.o ¥
boundary_cell.o cell_metrics.o ¥
input.o main.o poi_gen.o pointer_init.o outucd.o

TARGET = ../run/sol20

all: $(TARGET)
$(TARGET): $(OBJS)
        $(F90) $(F90FLAGS) -o $(TARGET) ¥
        $(OBJS) ¥
        $(F90FLAGS)
clean:
        rm -f *.o $(TARGET) *.mod *~ PI*
```

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

1 Node of Reedbush-U consists of 2 CPU's (Sockets)

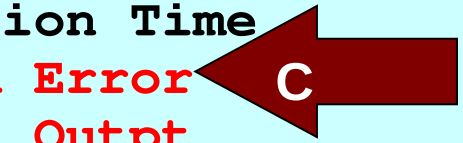


Only a single socket will be used in this class !

Job Script (1/2) go1.sh

- **<$O-omp>/run/go1.sh**
- Scheduling + Shell Script

```
#!/bin/sh
#PBS -q u-lecture7                    Name of "QUEUE"
#PBS -N test                          Job Name
#PBS -l select=1:ncpus=18             node#, core# (1-18)
#PBS -Wgroup_list=gt27                Group Name (Wallet)
#PBS -l walltime=00:05:00             Computation Time
#PBS -e test.err                      Standard Error
#PBS -o test.lst                      Standard Outpt

cd $PBS_O_WORKDIR                     go to current dir
. /etc/profile.d/modules.sh           (ESSENTIAL)

export OMP_NUM_THREADS=18             Thread# (=ncpus, 1-18)
export KMP_AFFINITY=granularity=fine,compact
numactl ./sol20                       execution
                                      (numactl:essential)
```

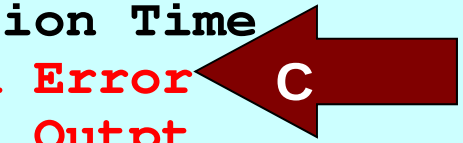


export KMP_AFFINITY=granularity=fine,compact
All of 1-18 threads are on Socket #0

Job Script (2/2) go2.sh

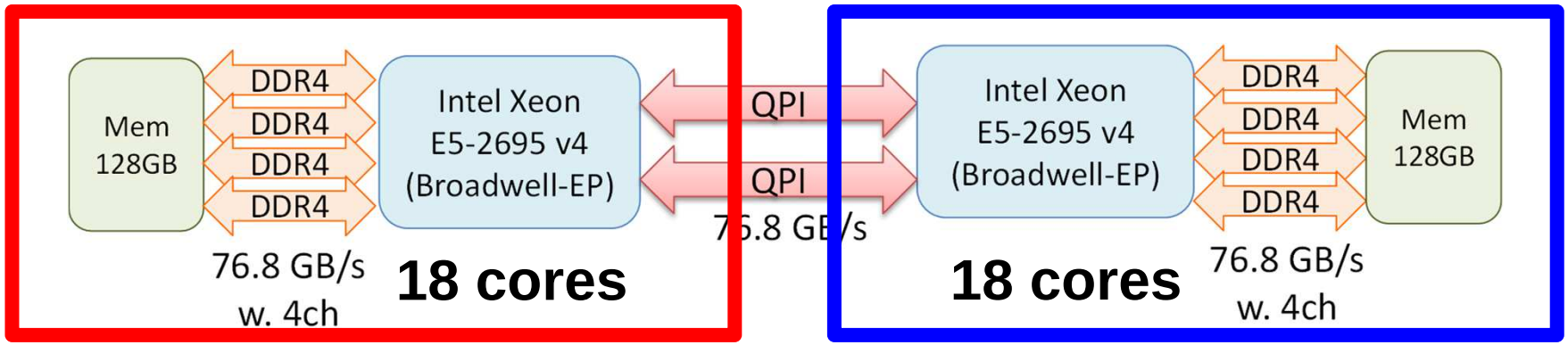
- **<$O-omp>/run/go2.sh**
- Scheduling + Shell Script

```
#!/bin/sh
#PBS -q u-lecture7                  Name of "QUEUE"
#PBS -N test                        Job Name
#PBS -l select=1:ncpus=18           node#, core# (1-18)
#PBS -Wgroup_list=gt27              Group Name (Wallet)
#PBS -l walltime=00:05:00           Computation Time
#PBS -e test.err                    Standard Error
#PBS -o test.lst                    Standard Outpt

cd $PBS_O_WORKDIR                   go to current dir
. /etc/profile.d/modules.sh         (ESSENTIAL)

export OMP_NUM_THREADS=18           Thread# (=ncpus, 1-18)

numactl ./sol20                     execution
                                    (numactl:essential)
```



Cores are randomly selected from Socket #0 & #1

1 Node of Reedbush-U 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

```
>$ cdw
>$ cd multicore/omp/run
(modify go1.sh)
>$ qsub go1.sh

>$ cat test.lst
```

INPUT.DAT

```
100 100 100                          NX NY NZ
1.00e-0  1.00e-00 1.00e-00           DX/DY/DZ
1.0e-08                              EPSICCG
```

Available QUEUE's

- Following 2 queues are available.
- 8 nodes can be used
  - **`u-lecture`**
    - 8 nodes (288 cores), 10 min., valid until the end of September, 2018
    - Shared by all "educational" users
  - **`u-lecture7`**
    - 4 nodes (144 cores), 10 min., active during class time
    - More jobs (compared to **`lecture`**) can be processed up on availability.

Submitting & Checking Jobs

• Submitting Jobs	`qsub SCRIPT NAME`
• Checking status of jobs	`rbstat`
• Deleting/aborting	`qdel JOB ID`
• Checking status of queues	`rbstat --rsc`
• Detailed info. of queues	`rbstat --rsc -x`
• Number of running jobs	`rbstat -b`
• History of Submission	`rbstat -H`
• Limitation of submission	`rbstat --limit`

Original: solve_PCG (1/3)

```
      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
...
      ITR= N

      do L= 1, ITR
!C
!C +---------------+
!C | {z}= [Minv]{r} |
!C +---------------+
!C===
      do i= 1, N
        W(i,Z)= W(i,R)*W(i,DD)
      enddo
!C===

!C
!C +-------------+
!C | RHO= {r}{z} |
!C +-------------+
!C===
      RHO= 0.d0
      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo
!C===
```

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

Original: solve_PCG (2/3)

```
!C
!C +-----------------------------+
!C | {p} = {z} if      ITER=1    |
!C | BETA= RHO / RHO1  otherwise |
!C +-----------------------------+
!C===
      if ( L.eq.1 ) then
        do i= 1, N
          W(i,P)= W(i,Z)
        enddo
       else
        BETA= RHO / RHO1
        do i= 1, N
          W(i,P)= W(i,Z) + BETA*W(i,P)
        enddo
      endif
!C===

!C
!C +------------+
!C | {q}= [A]{p} |
!C +------------+
!C===
      do i= 1, N
        VAL= D(i)*W(i,P)
        do k= indexL(i-1)+1, indexL(i)
          VAL= VAL + AL(k)*W(itemL(k),P)
        enddo
        do k= indexU(i-1)+1, indexU(i)
          VAL= VAL + AU(k)*W(itemU(k),P)
        enddo
        W(i,Q)= VAL
      enddo
!C===
```

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**
  - $\mathbf{p^{(1)}= z^{(0)}}$
  - **else**
    - $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    - $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  - **endif**
- $\mathbf{q^{(i)}= [A]p^{(i)}}$
- $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
- $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
- $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
- check convergence $|r|$

end

Original: solve_PCG (3/3)

```
!C
!C +--------------------+
!C | ALPHA= RHO / {p}{q} |
!C +--------------------+
!C===
      C1= 0.d0
      do i= 1, N
        C1= C1 + W(i,P)*W(i,Q)
      enddo
      ALPHA= RHO / C1
!C===
!C +--------------------+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +--------------------+
!C===
      do i= 1, N
        X(i)  = X(i)   + ALPHA * W(i,P)
        W(i,R)= W(i,R) - ALPHA * W(i,Q)
      enddo
      DNRM2= 0.d0
      do i= 1, N
        DNRM2= DNRM2 + W(i,R)**2
      enddo
!C===
      ERR = dsqrt(DNRM2/BNRM2)
      if (ERR .lt. EPS) then
        IER = 0
        goto 900
       else
        RHO1 = RHO
      endif

      enddo
      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

solve_PCG (1/5)

parallel computing by OpenMP

```
      module solver_PCG
      contains

      subroutine solve_PCG                                              &
     &         ( N, NPL, NPU, indexL, itemL, indexU, itemU, D, B, X,    &
     &           AL, AU, EPS, ITR, IER, N2)

      use omp_lib
      implicit REAL*8 (A-H,O-Z)
      integer :: N, NL, NU, N2

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

      real(kind=8), dimension(NPL) :: AL
      real(kind=8), dimension(NPU) :: AU

      integer, dimension(0:N)   :: indexL, indexU
      integer, dimension(NPL):: itemL
      integer, dimension(NPU):: itemU

      real(kind=8), dimension(:,:), allocatable :: W

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

solve_PCG (2/5)

```
!$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,VAL,j)
      do i= 1, N
        VAL= D(i)*X(i)
        do k= indexL(i-1)+1, indexL(i)
          VAL= VAL + AL(k)*X(itemL(k))
        enddo
        do k= indexU(i-1)+1, indexU(i)
          VAL= VAL + AU(k)*X(itemU(k))
         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
```

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 (3/5)

```
      ITR= N
      Stime= omp_get_wtime()

      do L= 1, ITR

!$omp parallel do private(i)
      do i= 1, N
        W(i,Z)= W(i,R)*W(i,DD)
      enddo

      RHO= 0.d0
!$omp parallel do private(i) reduction(+:RHO)
      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo

      if ( L.eq.1 ) then
!$omp parallel do private(i)
        do i= 1, N
          W(i,P)= W(i,Z)
        enddo
       else
        BETA= RHO / RHO1
!$omp parallel do private(i)
        do i= 1, N
          W(i,P)= W(i,Z) + BETA*W(i,P)
        enddo
      endif
```

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)

```
!$omp parallel do private(i,VAL,j)
      do i= 1, N
        VAL= D(i)*W(i,P)
        do k= indexL(i-1)+1, indexL(i)
          VAL= VAL + AL(k)*W(itemL(k),P)
        enddo
        do k= indexU(i-1)+1, indexU(i)
          VAL= VAL + AU(k)*W(itemU(k),P)
        enddo
        W(i,Q)= VAL
      enddo

      C1= 0.d0
!$omp parallel do private(i) reduction(+:C1)
      do i= 1, N
        C1= C1 + W(i,P)*W(i,Q)
      enddo

      ALPHA= RHO / C1

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

      DNRM2= 0.d0
!$omp parallel do private(ip,i) reduction(+:DNRM2)
      do i= 1, N
        DNRM2= DNRM2 + W(i,R)**2
      enddo

      ERR = dsqrt(DNRM2/BNRM2)...
```

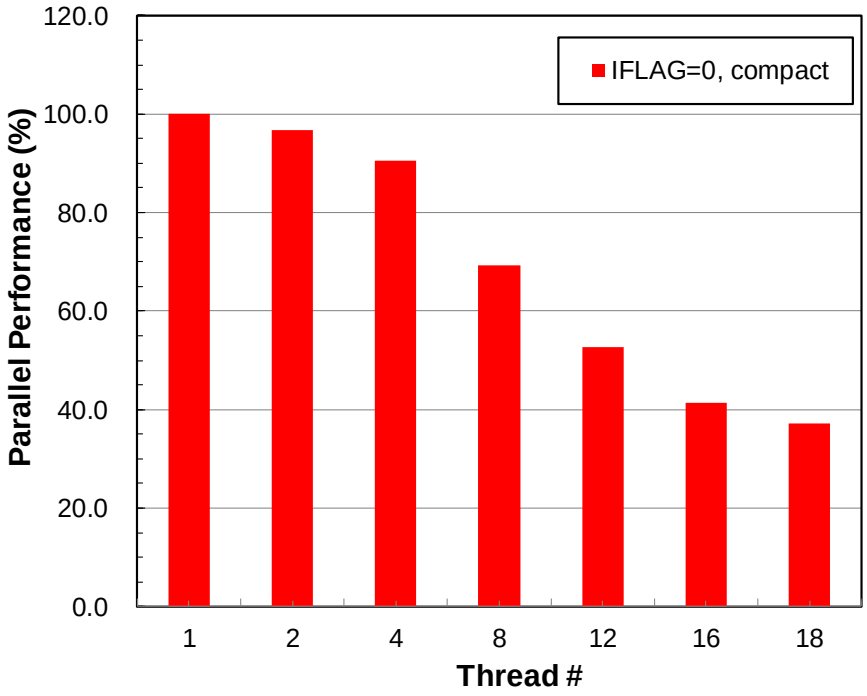
Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$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

solve_PCG (5/5)

```
      Stime = omp_get_wtime()

      do L= 1, ITR
...
        if (ERR .lt. EPS) then
          IER = 0
          goto 900
         else
          RHO1 = RHO
        endif

      enddo
      IER = 1
  900 continue
      Etime= omp_get_wtime()

      write (*,'(i5,2(1pe16.6))') L, ERR
      write (*,'(1pe16.6, a)') Etime-Stime, ' sec. (solver)'

      ITR= L
      deallocate (W)

      return
      end
```

Elapsed Time= Etime - Stime

Results: Parallel Performance NX=NY=NZ=100: go1.sh

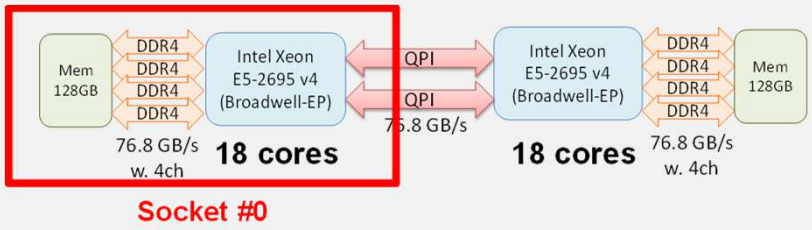


Thread #	sec	Speed-up
1	10.219	1.000
2	5.277	1.936
4	2.824	3.618
8	1.842	5.547
12	1.616	6.322
16	1.542	6.628
18	1.530	6.679

go1.sh

Only cores on a single socket used

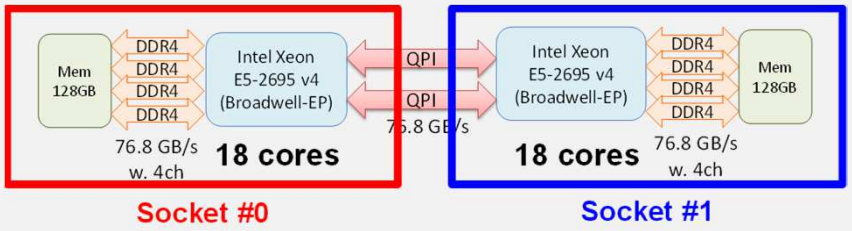
```
#!/bin/sh

#PBS -q u-lecture7
#PBS -N test01
#PBS -l select=1:ncpus=18      1,2,4,8,12,16,18
#PBS -Wgroup_list=gt27
#PBS -l walltime=00:10:00
#PBS -e test1.err
#PBS -o test1.lst

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh
export KMP_AFFINITY=granularity=fine,compact

export OMP_NUM_THREADS=18      1,2,4,8,12,16,18
numactl ./sol20
```



go2.sh

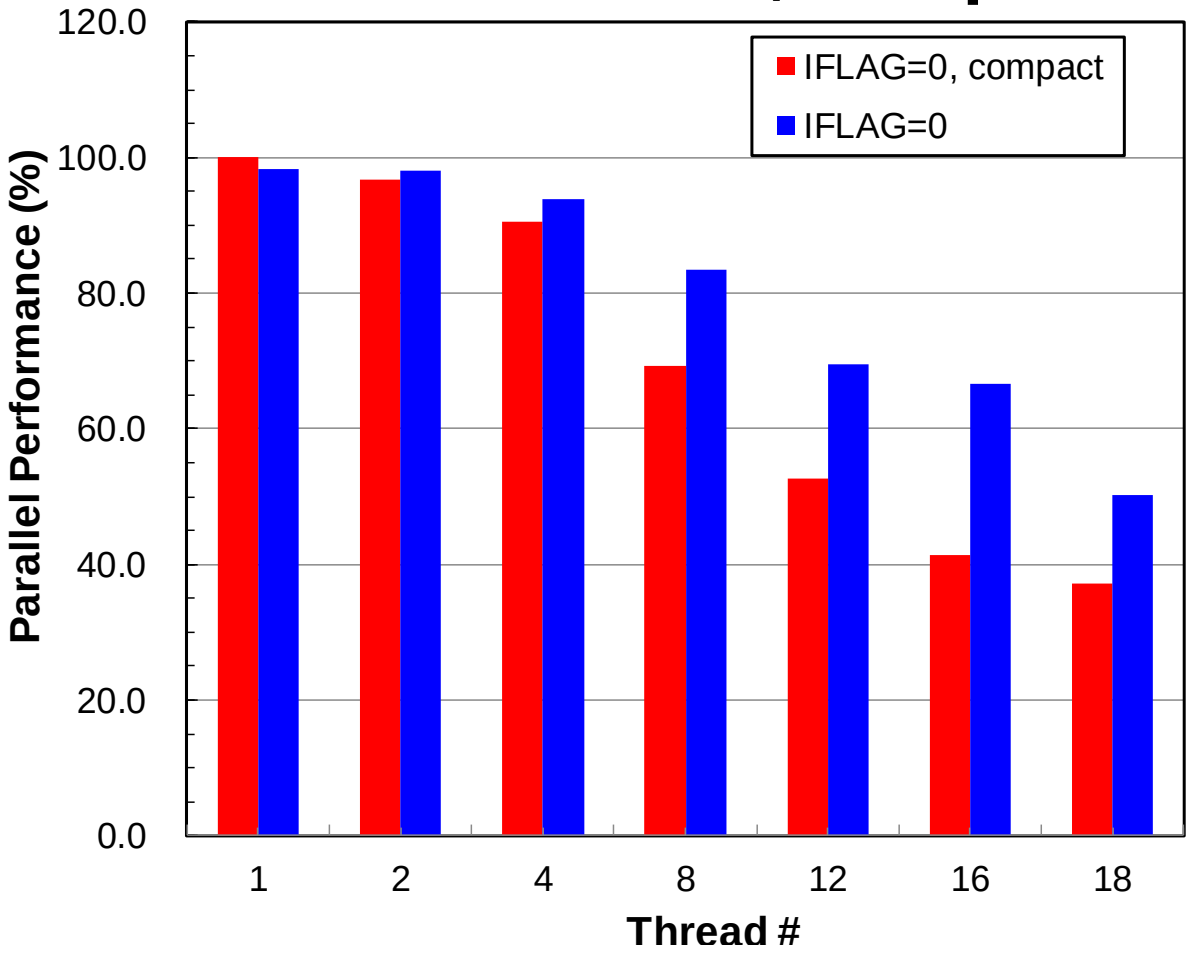
cores are randomly selected from 2 sockets

```
#!/bin/sh

#PBS -q u-lecture7
#PBS -N test01
#PBS -l select=1:ncpus=18      1,2,4,8,12,16,18
#PBS -Wgroup_list=gt27
#PBS -l walltime=00:10:00
#PBS -e test2.err
#PBS -o test2.lst

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh


export OMP_NUM_THREADS=18      1,2,4,8,12,16,18
numactl ./sol20
```



Results: Parallel Performance

NX=NY=NZ=100, IFLAG=0
Measurement: 5 times, best case
go2.sh is better if Thread # is more than 8
based on “IFLAG=0, compact” with 1 thread



go1.sh

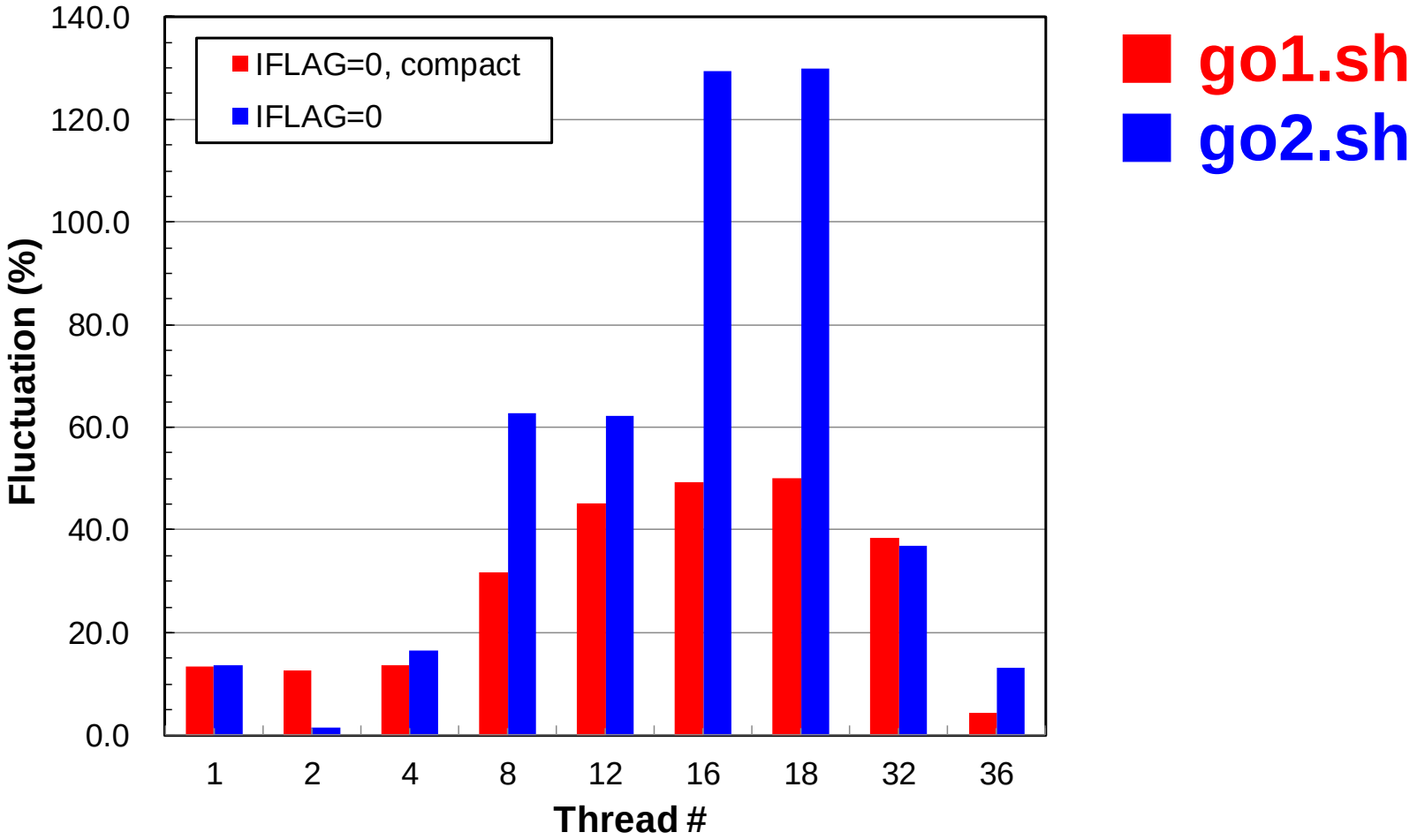
go2.sh

Results: Fluctuation Rate

NX=NY=NZ=100, IFLAG=0

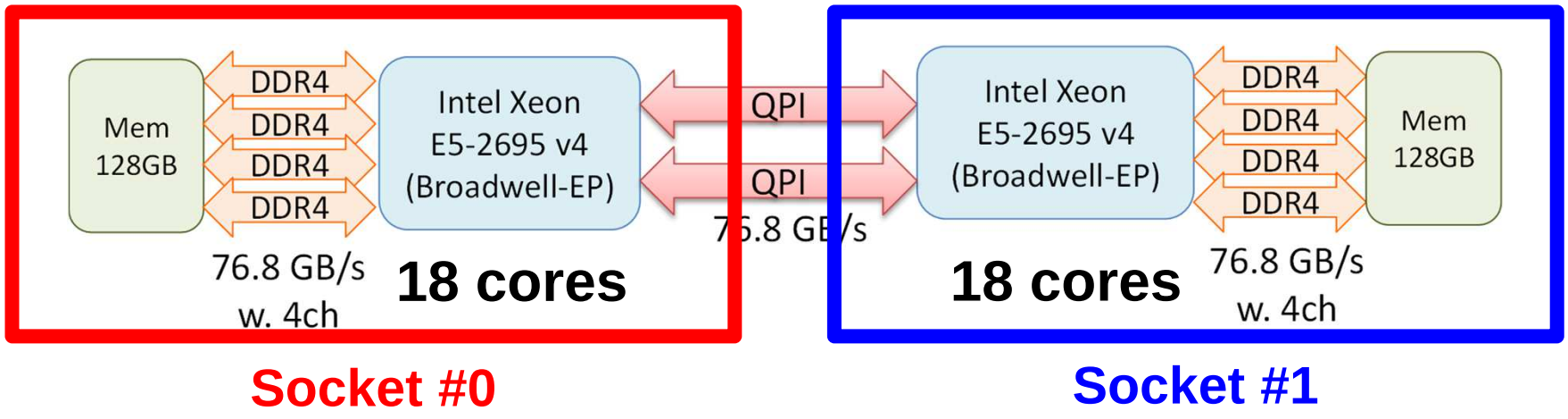
Measurement: 5 times, (Worst-Best)/Best

Core Allocation of go2.sh is random



Reedbush-U

1-node: 2-CPU's/sockets



- Each Node of Reedbush-U
  - 2 Sockets (CPU's) of Intel Broadwell-EP
  - Each socket has 18 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/socket could be smaller -> more efficient
  - But access to remote memory may happen -> NUMA, to be taught later

Exercises

- Effect of problem size (NX, NY, NZ)
- Effect of Thread # (OMP_NUM_THREADS: 1-18)

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

Why less than 18x ?

- 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 larger

Sparse/Dense Matrices

```
do i= 1, N
  Y(i)= D(i)*X(i)
  do k= index(i-1)+1, index(i)
    Y(i)= Y(i) + AMAT(k)*X(item(k))
  enddo
enddo
```

```
do j= 1, N
  do i= 1, N
    Y(j)= Y(j) + A(i,j)*X(i)
  enddo
enddo
```

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

GeoFEM Benchmark

ICCG in FEM for Solid Mechanics

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

Sparse Linear Solver: Memory-Bound

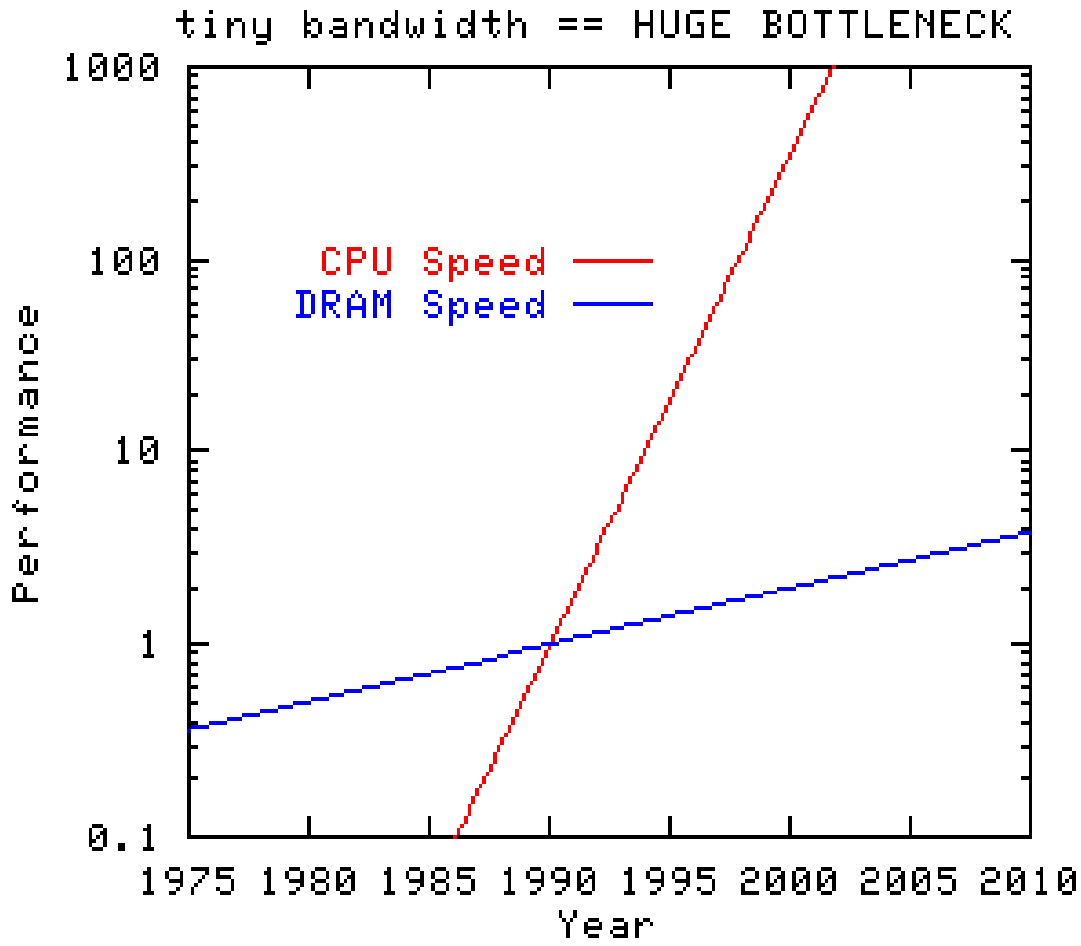
STREAM benchmark

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

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

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

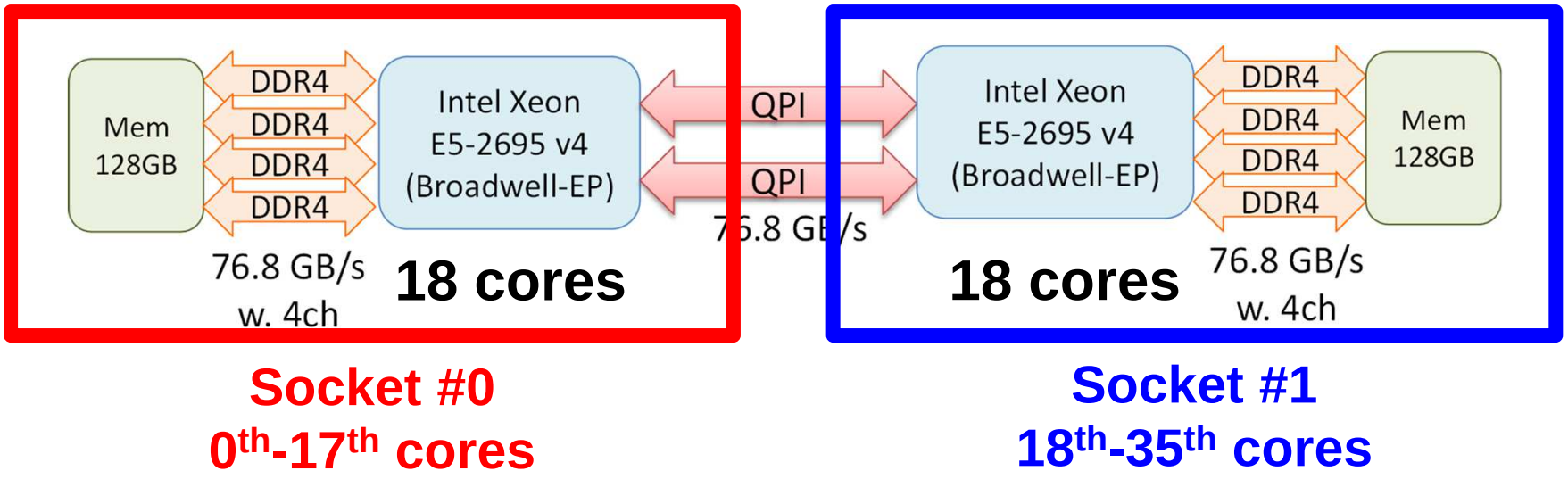
Gap between performance of CPU and Memory



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

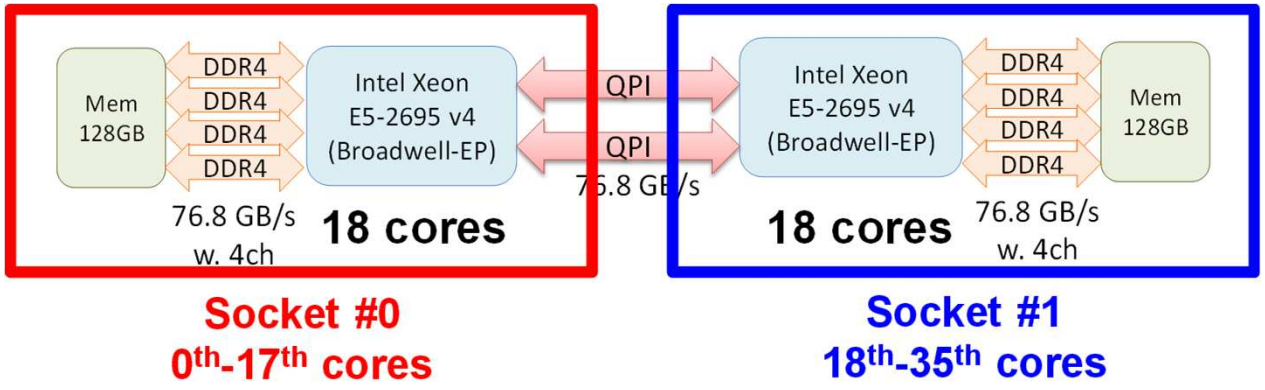
How to run: MPI Version

```
>$ cdw
>$ cd multicore/stream
>$ mpiifort -O3 -xCORE-AVX2 -align array32byte stream.f -o stream

>$ qsub XXX.sh
```



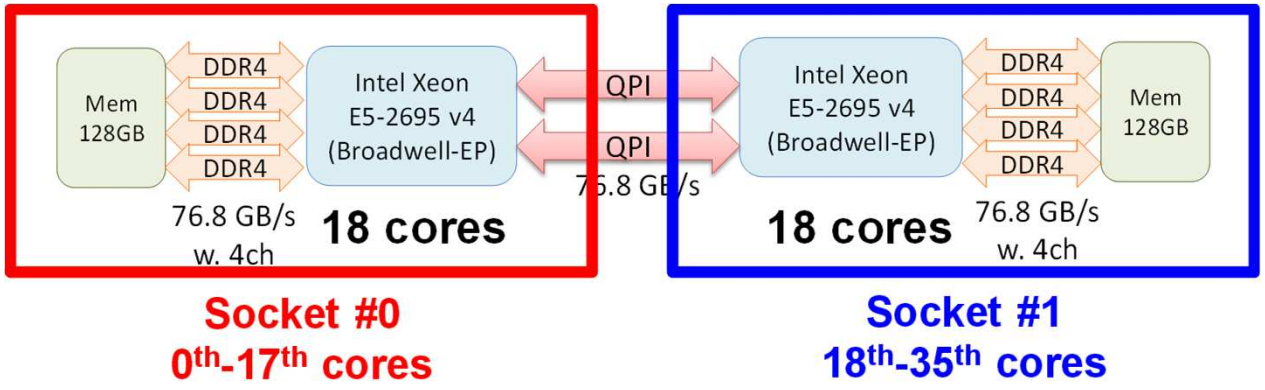
s01.sh: Use 1 core (0th)

```
#!/bin/sh

#PBS -q u-lecture7
#PBS -N stream
#PBS -l select=1:mpiprocs=1      MPI Process # (1-36)
#PBS -Wgroup_list=gt27
#PBS -l walltime=00:05:00
#PBS -e err
#PBS -o t01.lst

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

export I_MPI_PIN_PROCESSOR_LIST=0      use 0th core
mpirun ./impimap.sh ./stream
```



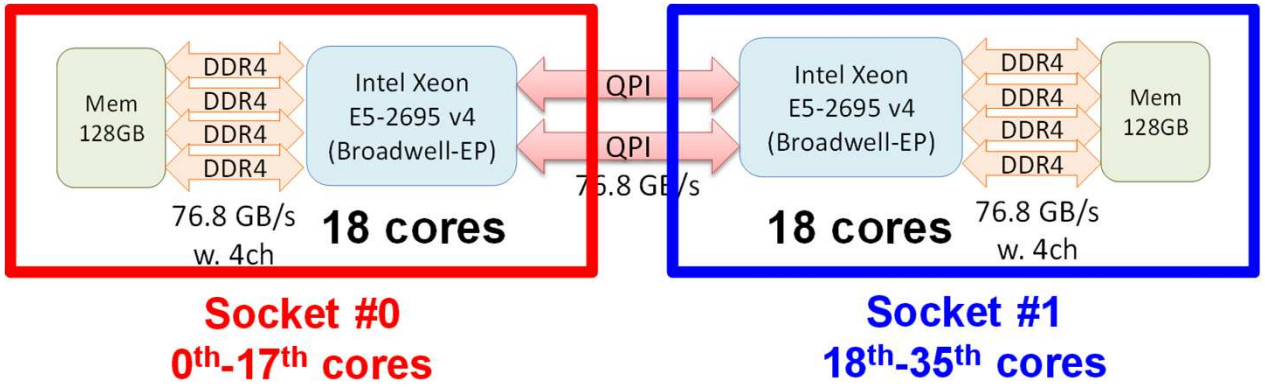
s16.sh: Use 16 cores (0-15th)

```
#!/bin/sh

#PBS -q u-lecture7
#PBS -N stream
#PBS -l select=1:mpiprocs=16     MPI Process # (1-36)
#PBS -Wgroup_list=gt27
#PBS -l walltime=00:05:00
#PBS -e err
#PBS -o t16.lst

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

export I_MPI_PIN_PROCESSOR_LIST=0-15   use 0-15th core
mpirun ./impimap.sh ./stream
```



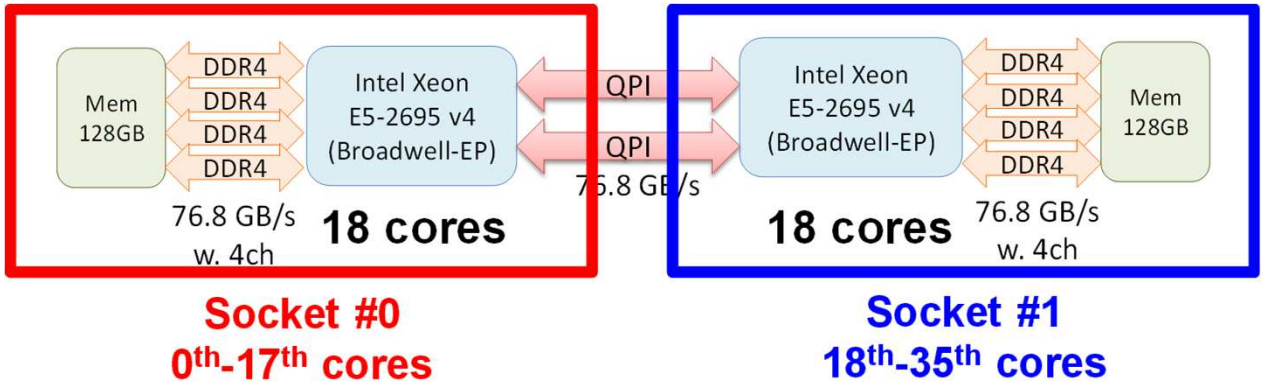
s32.sh: Use 32 cores (16 ea)

```
#!/bin/sh

#PBS -q u-lecture7
#PBS -N stream
#PBS -l select=1:mpiprocs=32     MPI Process # (1-36)
#PBS -Wgroup_list=gt27
#PBS -l walltime=00:05:00
#PBS -e err
#PBS -o t32.lst

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

export I_MPI_PIN_PROCESSOR_LIST=0-15,18-33
mpirun ./impimap.sh ./stream
```



s36.sh: Use 36 cores (ALL)

```
#!/bin/sh

#PBS -q u-lecture7
#PBS -N stream
#PBS -l select=1:mpiprocs=36     MPI Process # (1-36)
#PBS -Wgroup_list=gt27
#PBS -l walltime=00:05:00
#PBS -e err
#PBS -o t36.lst

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

export I_MPI_PIN_PROCESSOR_LIST=0-35
mpirun ./impimap.sh ./stream
```



Results of Triad on a Single Node of Reedbush-U, Peak is 153.6 GB/sec.

Thread #	GB/sec	Speed-up
1	16.11	1.00
2	30.95	1.92
4	54.72	3.40
8	64.59	4.01
16	65.18	4.04
18	64.97	4.03
32	130.0	8.07
36	128.2	7.96

Exercises

- Running the code
- Try various number of threads (1-36)
  - export I_MPI_PIN_PROCESSOR_LIST=0-7
  - export I_MPI_PIN_PROCESSOR_LIST=0-3,18-21
- OpenMP-version and Single PE version are available
  - Fortran, C
  - http://www.cs.virginia.edu/stream/

- OpenMP
- Login to Reedbush-U
- 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

```
do i= 1, N
  W(i,Z)= W(i,R)*W(i,DD)
enddo
```

```
!$omp parallel do private(i)
  do i = 1, N
    W(i,Z)= W(i,R)*W(i,DD)
  enddo
!$omp end parallel do
```

```
!$omp parallel do private(ip,i)
  do ip= 1, PEsmpTOT
  do i = INDEX(ip-1)+1, INDEX(ip)
    W(i,Z)= W(i,R)*W(i,DD)
  enddo
  enddo
!$omp end parallel do
```

```
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

```
do i= 1, N
  VAL= D(i)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL - (AL(k)**2) * W(itemL(k),DD)
  enddo
  W(i,DD)= 1.d0/VAL
enddo
```

Forward Substitution

```
do i= 1, N
  WVAL= W(i,Z)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL -  AL(k) * W(itemL(k),Z)
  enddo
  W(i,Z)= WVAL * W(i,DD)
enddo
```

Data Dependency

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

IC Factorization

```
do i= 1, N
  VAL= D(i)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL - (AL(k)**2) * W(itemL(k),DD)
  enddo
  W(i,DD)= 1.d0/VAL
enddo
```

Forward Substitution

```
do i= 1, N
  WVAL= W(i,Z)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL -  AL(k) * W(itemL(k),Z)
  enddo
  W(i,Z)= WVAL * W(i,DD)
enddo
```

Forward Substitution
Four Thread Parallel Operation

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

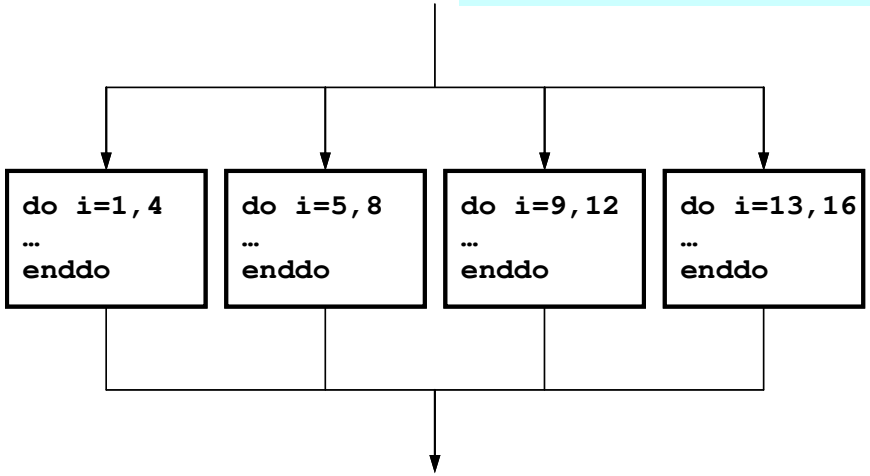
```
do i= 1, N
  WVAL= W(i,Z)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL -  AL(k) * W(itemL(k),Z)
  enddo
  W(i,Z)= WVAL * W(i,DD)
enddo
```

Forward Substitution
Four Thread Parallel Operation

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

```
!$omp parallel do private (ip,i,k,VAL)
  do ip= 1, 4
  do  i= INDEX(ip-1)+1, INDEX(ip)
    WVAL= W(i,Z)
    do k= indexL(i-1)+1, indexL(i)
      WVAL= WVAL -  AL(k) * W(itemL(k),Z)
    enddo
    W(i,Z)= WVAL * W(i,DD)
  enddo
  enddo
!$omp parallel enddo
```

```
INDEX(0)= 0
INDEX(1)= 4
INDEX(2)= 8
INDEX(3)=12
INDEX(4)=16
```



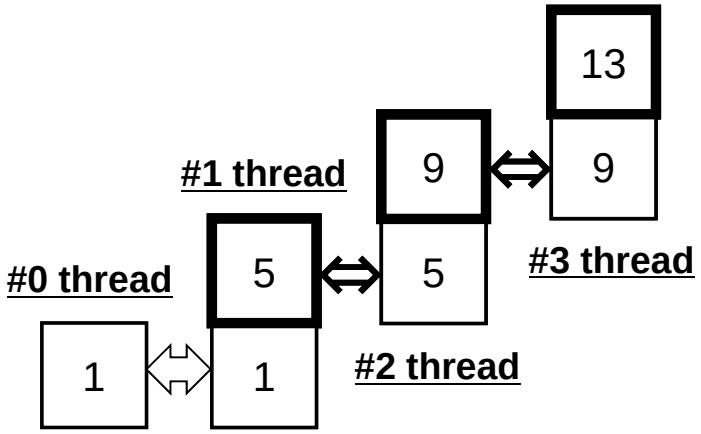
These four threads are executed simultaneously.

Data Dependency: Conflict of reading from/writing to memory

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

```
!$omp parallel do private (ip,I,k,VAL)
  do ip= 1, 4
  do  i= INDEX(ip-1)+1, INDEX(ip)
    WVAL= W(i,Z)
    do k= indexL(i-1)+1, indexL(i)
      WVAL= WVAL -  AL(k) * W(itemL(k),Z)
    enddo
    W(i,Z)= WVAL * W(i,DD)
  enddo
  enddo
!$omp parallel enddo
```

```
INDEX(0)= 0
INDEX(1)= 4
INDEX(2)= 8
INDEX(3)=12
INDEX(4)=16
```



⇔:
Data Dependency

Parallelize ICCG Method

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