OpenMPによるマルチコア・メニィコア並列プログラミング入門 C言語編

Part-A2：OpenMP

中島研吾

東京大学情報基盤センター

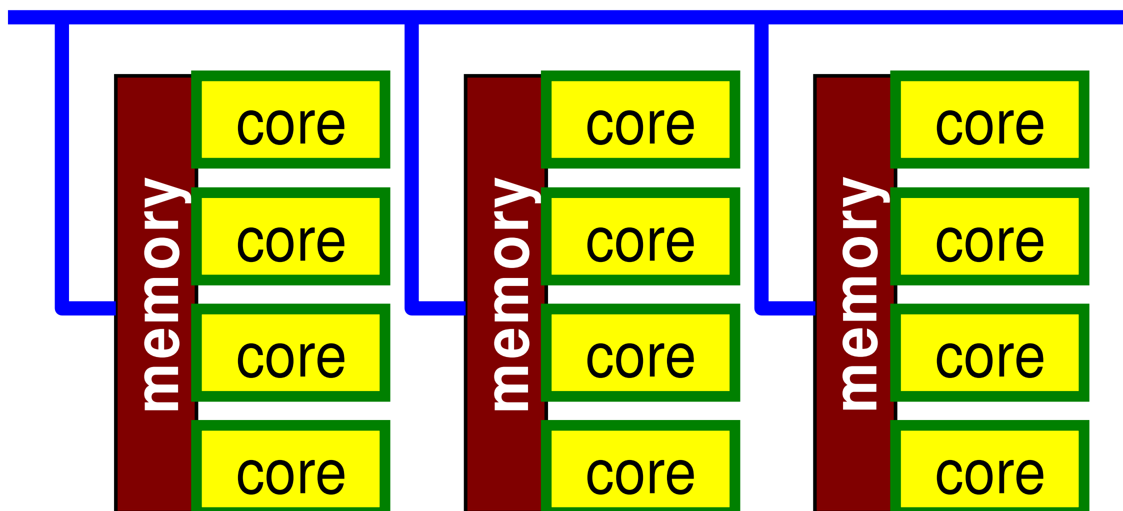
- OpenMP
- Login to Wisteria/BDEC-01
- OpenMPによる並列化（0）（12コアまで）
- OpenMPによる並列化（1）（First Touch）
- OpenMPによる並列化（2）（＋ELL）
- OpenMPによる並列化（3）（＋omp-parallel削減）
- OpenMPによる並列化（4）（＋更なる最適化（Fortranのみ））

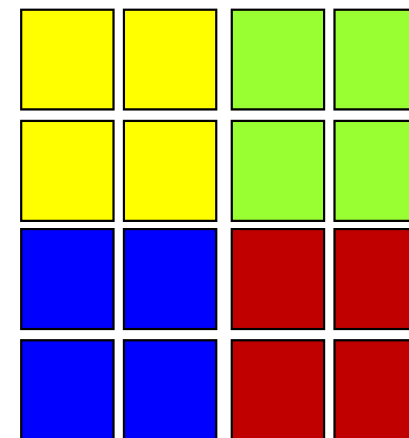
Hybrid並列プログラミング

- スレッド並列＋メッセージパッシング
  - OpenMP+ MPI
  - CUDA + MPI, OpenACC + MPI
- OpenMPがMPIより簡単ということはない
  - データ依存性のない計算であれば，機械的にOpenMP指示文を入れれば良い
  - NUMAになるとより複雑：First Touch Data Placement

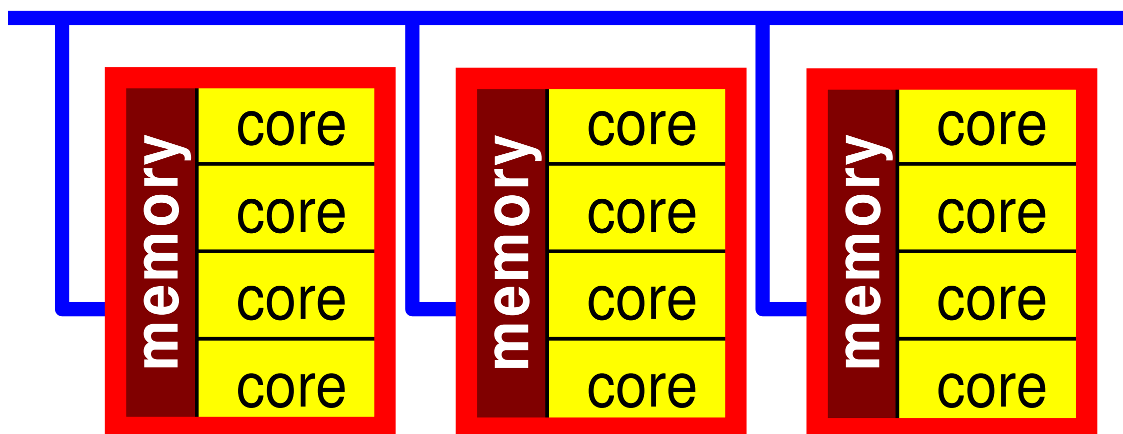
Flat MPI vs. Hybrid

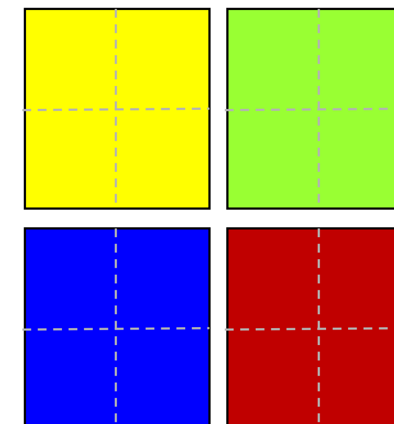
Flat-MPI：Each Core -> Independent



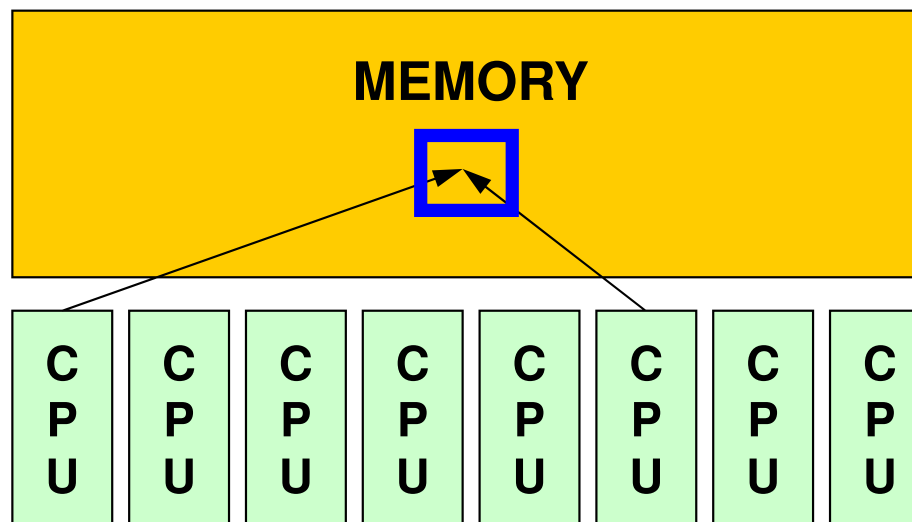


Hybrid：Hierarchal Structure





共有メモリ型計算機



- SMP
  - Symmetric Multi Processors
  - 複数のCPU（コア）で同じメモリ空間を共有するアーキテクチュア

OpenMPとは（1/2）

http://www.openmp.org

- 共有メモリ型並列計算機用のDirectiveの統一規格
  - MPIやHPFに比べると遅く1996年頃から活動開始
  - 現在 Ver.4.X
- 背景
  - CrayとSGIの合併（1996）
  - ASCI計画の開始（1995）
    - Accelerated Strategic Computing Initiative (ASCI) -> Advanced Simulation and Computing Program (ASC)
    - ASCI：核実験のシミュレーションによる代替
      - 計算機開発，シミュレーションソフトウェア
    - SMPクラスタにフォーカス
      - ASCI Red (Intel), Blue Pacific/White/Purple/BlueGene (IBM), Blue Mountain（SGI）
    - SMPクラスタ向けの並列プログラミングの共通API（Application Program Interface）が必要

OpenMPとは（2/2）

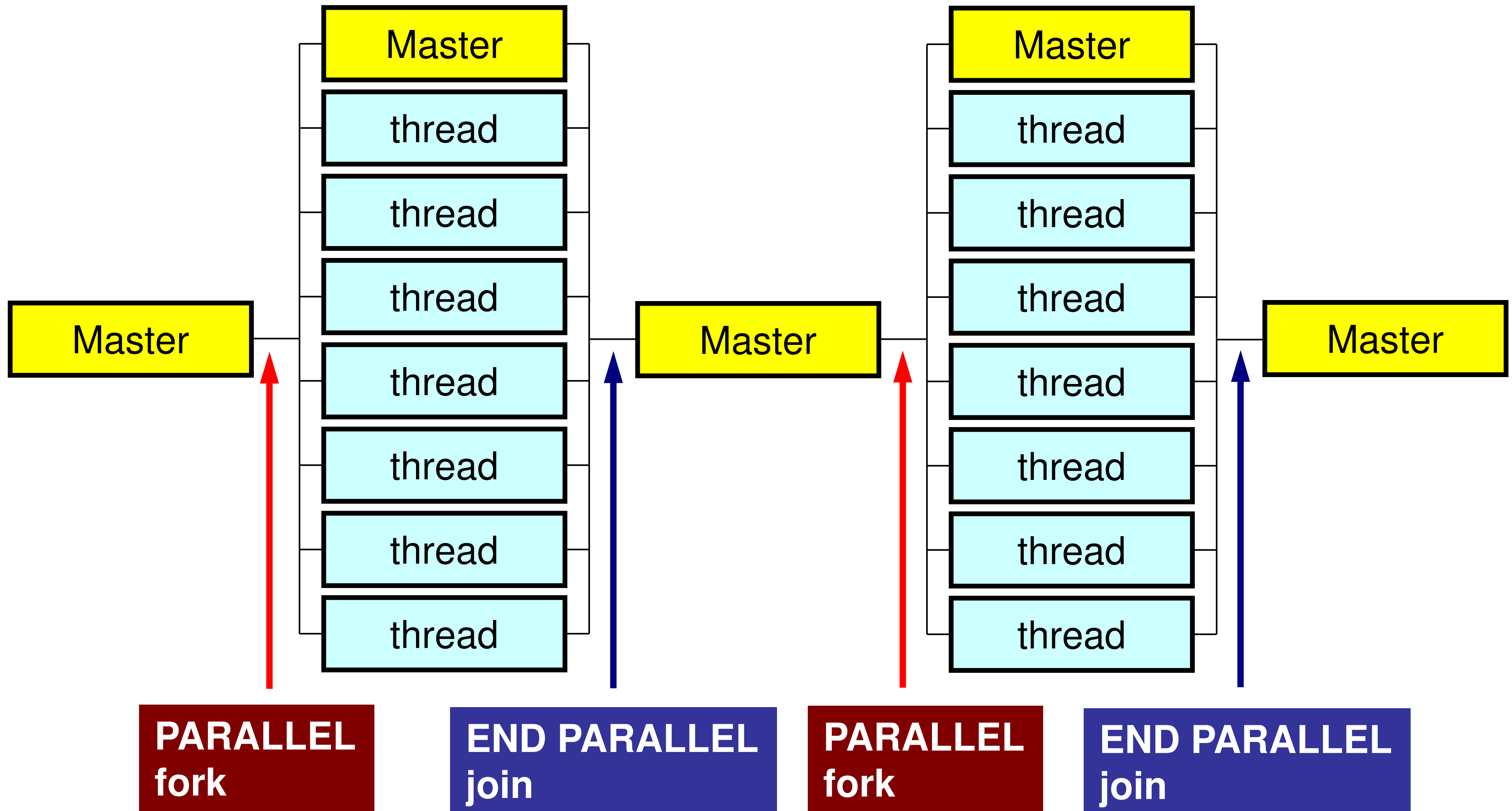
http://www.openmp.org

- 主な計算機ベンダーが集まって OpenMP ARBを結成し，1997年にはもう規格案ができていたそうである
  - SC98ではすでにOpenMPのチュートリアルがあったし，すでにSGI Origin2000でOpenMP-MPIハイブリッドのシミュレーションをやっている例もあった。
- OpenMPはFortan版とC/C++版の規格が全く別々に進められてきた。
  - Ver.2.5で言語間の仕様を統一
- Ver.4.0ではGPU，Intel-MIC等Co-Processor，Accelerator環境での動作も考慮
  - OpenACCに近づいている

OpenMPの概要

- 基本的仕様
  - プログラムを並列に実行するための動作をユーザーが明示
  - OpenMP実行環境は，依存関係，衝突，デッドロック，競合条件，結果としてプログラムが誤った実行につながるような問題に関するチェックは要求されていない。
  - プログラムが正しく実行されるよう構成するのはユーザーの責任である。
- 実行モデル
  - fork-join型並列モデル
    - 当初はマスタスレッドと呼ばれる単一プログラムとして実行を開始し，「PARALLEL」，「END PARALLEL」ディレクティヴの対で並列構造を構成する。並列構造が現れるとマスタスレッドはスレッドのチームを生成し，そのチームのマスタとなる。
  - いわゆる「入れ子構造」も可能であるが，ここでは扱わない

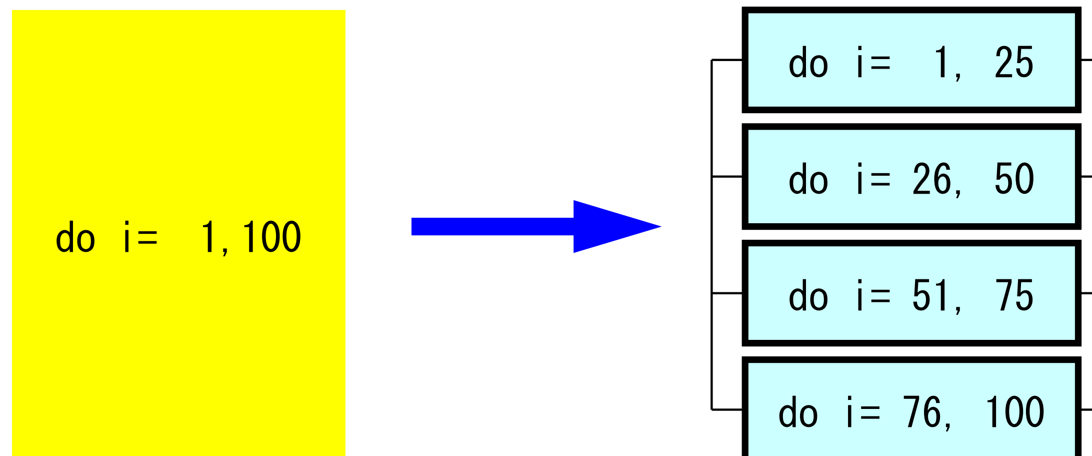
Fork-Join 型並列モデル



スレッド数

- 環境変数 **OMP_NUM_THREADS**
  - 値の変え方
    - bash(.bashrc) **export OMP_NUM_THREADS=8**
    - csh(.cshrc) **setenv OMP_NUM_THREADS 8**

- たとえば，**OMP_NUM_THREADS=4**とすると，以下のように**i=1~100**のループが**4**分割され，同時に実行される。



OpenMPに関連する情報

- OpenMP Architecture Review Board (ARB)
  - http://www.openmp.org
- 参考文献
  - 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）
- 富士通他による翻訳：（OpenMP 3.0）必携！
  - http://www.openmp.org/mp-documents/OpenMP30spec-ja.pdf

OpenMPの特徴

- ディレクティヴ（指示行）の形で利用
  - 挿入直後のループが並列化される
  - コンパイラがサポートしていなければ，コメントとみなされる

OpenMP/Directives Array Operations

Simple Substitution

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

Dot Products

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

DAXPY

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

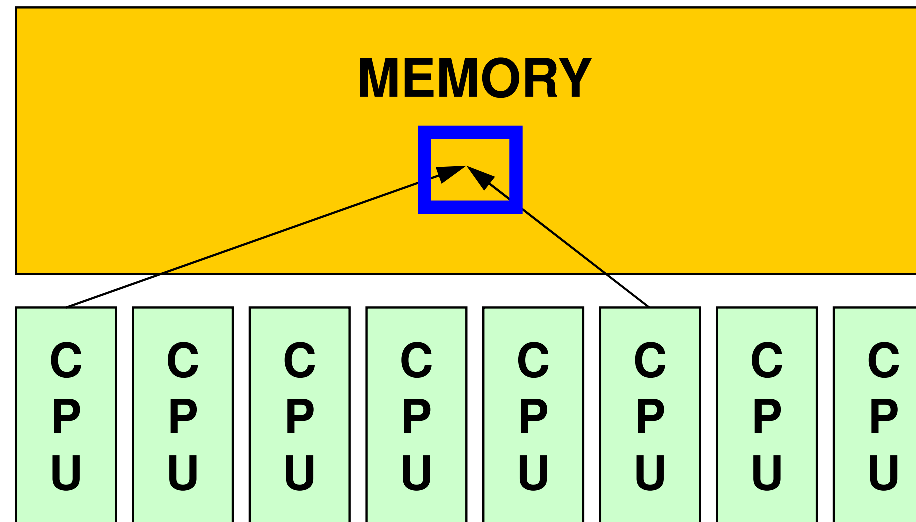
OpenMP/Direceives Matrix/Vector Products

```
#pragma omp parallel for private (i,VAL,j)
 for(i=0; i<N; i++) {

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

     W[Q][i] = VAL;
 }
```

OpenMPの特徴

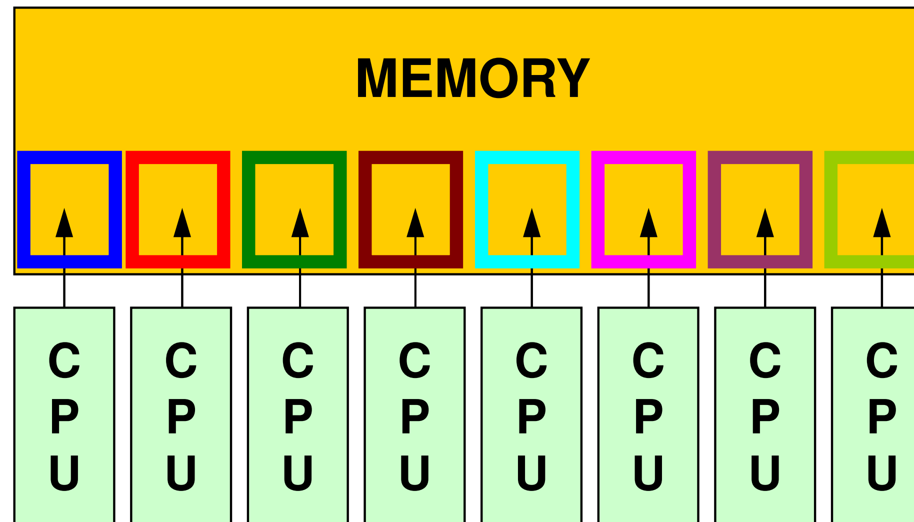
- ディレクティヴ（指示行）の形で利用
  - 挿入直後のループが並列化される
  - コンパイラがサポートしていなければ，コメントとみなされる
- 何も指定しなければ，何もしない
  - 「自動並列化」，「自動ベクトル化」とは異なる。
  - 下手なことをするとおかしな結果になる：ベクトル化と同じ
  - データ分散等（Ordering）は利用者の責任
- 共有メモリユニット内のプロセッサ数に応じて，「Thread」が立ち上がる
  - 「Thread」：MPIでいう「プロセス」に相当する。
  - 普通は「Thread数＝共有メモリユニット内プロセッサ数，コア数」であるが最近のアーキテクチャではHyper Threading（HT）がサポートされているものが多い（1コアで2-4スレッド）

メモリ競合（Memory Contention）



- 複雑な処理をしている場合，複数のスレッドがメモリ上の同じアドレスにあるデータを同時に更新する可能性がある。
  - 複数のCPUが配列の同じ成分を更新しようとする。
  - メモリを複数のコアで共有しているためこのようなことが起こりうる。
  - 場合によっては答えが変わる

メモリ競合（Memory Contention）（続き）



- 本演習で扱っている例は，そのようなことが生じないよう，各スレッドが同時に同じ成分を更新するようなことはないようにする（実はそもそも無い）。
  – これはユーザーの責任でやること，である。
- データ依存性（Data Dependency）
- コア数（スレッド数）が増えるほど，メモリへの負担が増えて，処理速度は低下する（メモリ飽和）。

OpenMPの特徴（続き）

- 基本は「#pragma omp parallel for」
- 変数について，グローバル/sharedな変数と，Thread内でローカルな「private」な変数に分けられる。
  - デフォルトは「global/shared」
  - 内積を求める場合は「reduction」を使う

```
     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];
        }
    }
```

W(:,:), R, Z, PEsmpTOT などはグローバル変数

FORTRANとC

```fortran
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
```

```c
#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];
}
```

本講義における方針

- OpenMPは多様な機能を持っているが，それらの全てを逐一教えることはしない。
  - 講演者も全てを把握，理解しているわけではない。

- 数値解析に必要な最低限の機能のみ学習する。
  - 具体的には，講義で扱っているPCG法によるポアソン方程式ソルバーを動かすために必要な機能のみについて学習する
  - それ以外の機能については，自習，質問のこと（全てに答えられるとは限らない）。

最初にやること

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

- 様々な環境変数，インタフェースの定義（OpenMP3.0以降でサポート）

OpenMPディレクィヴ（FORTRAN）

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

- 大文字小文字は区別されない。
- sentinel
  - 接頭辞
  - FORTRANでは「!$OMP」，「C$OMP」，「*$OMP」，但し自由ソース形式では「!$OMP」のみ。
  - 継続行にはFORTRANと同じルールが適用される。以下はいずれも「!$OMP PARALLEL DO SHARED(A,B,C)」

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

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

OpenMPディレクィヴ（C）

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

- 継続行は「＼」
- 小文字を使用（変数名以外）

```
#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)
```

- 多重スレッドによって実行される領域を定義し，DOループの並列化を実施する。
- 並び項目（clause）：よく利用するもの
  - private（list）
  - shared（list）
  - default（private|shared|none）
  - reduction（{operation|intrinsic}: list）

REDUCTION

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

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

- 「MPI_REDUCE」のようなものと思えばよい
- Operator
  - ＋, *, -, .AND., .OR., .EQV., .NEQV.
- Intrinsic
  - MAX, MIN, IAND, IOR, IEQR

実例A1：簡単なループ

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

- ループの繰り返し変数（ここでは「i」）はデフォルトでprivateなので，明示的に宣言は不要。
- 「END PARALLEL DO」は省略可能。
  - C言語ではそもそも存在しない

実例A2：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;
 }
```

OpenMP使用時に呼び出すことのできる関数群

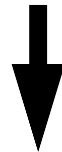
関数名	内容
int omp_get_num_threads (void)	スレッド総数
int omp_get_thread_num (void)	自スレッドのID
double omp_get_wtime (void)	MPI_Wtimeと同じ
void omp_set_num_threads (int num_threads) call omp_set_num_threads (num_threads)	スレッド数設定

OpenMP for Dot Products

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

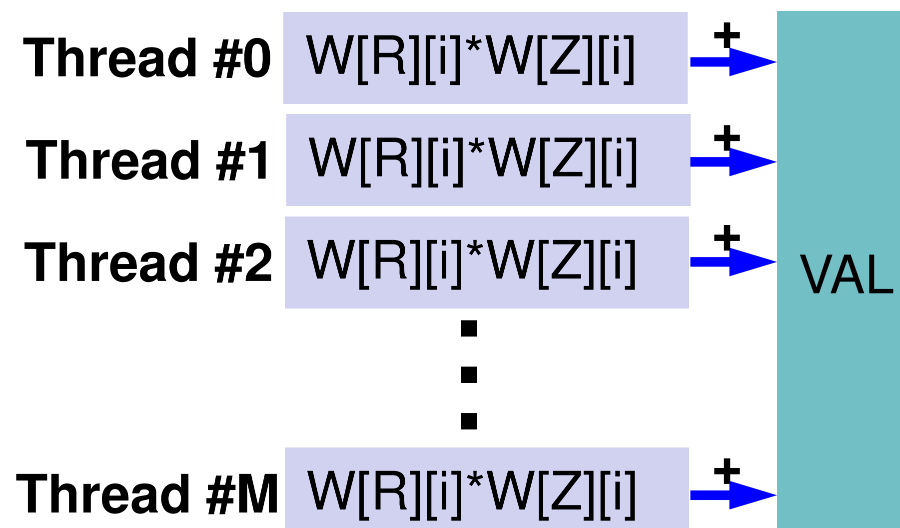
OpenMP for Dot Products

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



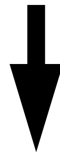
```
    VAL= 0.0;
#pragma omp parallel for private (i) reduction(+:VAL)
    for(i=0; i<N; i++){
      VAL= VAL + W[R][i] * W[Z][i];
    }
```

Directives are just inserted.



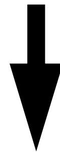
OpenMP for Dot Products

```
    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];
    }
```

OpenMPディレクティヴの挿入
これでも並列計算は可能

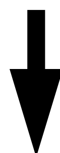


```
     VAL= 0.0;
#pragma omp parallel for private (i,ip)
 reduction(+:VAL)
    for(ip=0; ip<PEsmpTOT; ip++){
        for (i=INDEX[ip]; i<INDEX[ip+1]; i++){
           VAL= VAL + W[R][i] * W[Z][i];
        }
    }
```

多重ループの導入
PEsmpTOT：スレッド数
あらかじめ「INDEX(:)」を用意しておく
より確実に並列計算実施
（別に効率がよくなるわけでは無い）

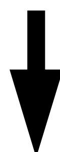
OpenMP for Dot Products

```
    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];
    }
```

OpenMPディレクティヴの挿入
これでも並列計算は可能



```
     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];
        }
    }
```

多重ループの導入
PEsmpTOT:スレッド数
あらかじめ「INDEX(:)」を用意しておく
より確実に並列計算実施

PEsmpTOT個のスレッドが立ち上がり，並列に実行

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];
        }
    }
```

多重ループの導入
PEsmpTOT：スレッド数
あらかじめ「INDEX[:]」を用意しておく
より確実に並列計算実施

PEsmpTOT個のスレッドが立ち上がり，並列に実行

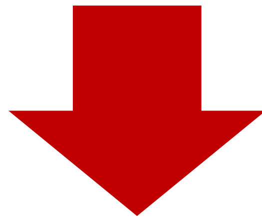
各要素が計算されるスレッドを指定できる

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

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

Matrix-Vector Multiply

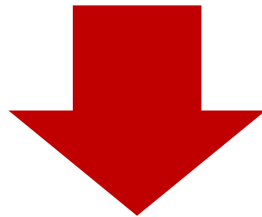
```
for(i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for(j=indexLU[i]; j<indexLU[i+1]; j++) {
        VAL += AMAT[j] * W[P][itemLU[j]];
    }
    W[Q][i] = VAL;
}
```



```
#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

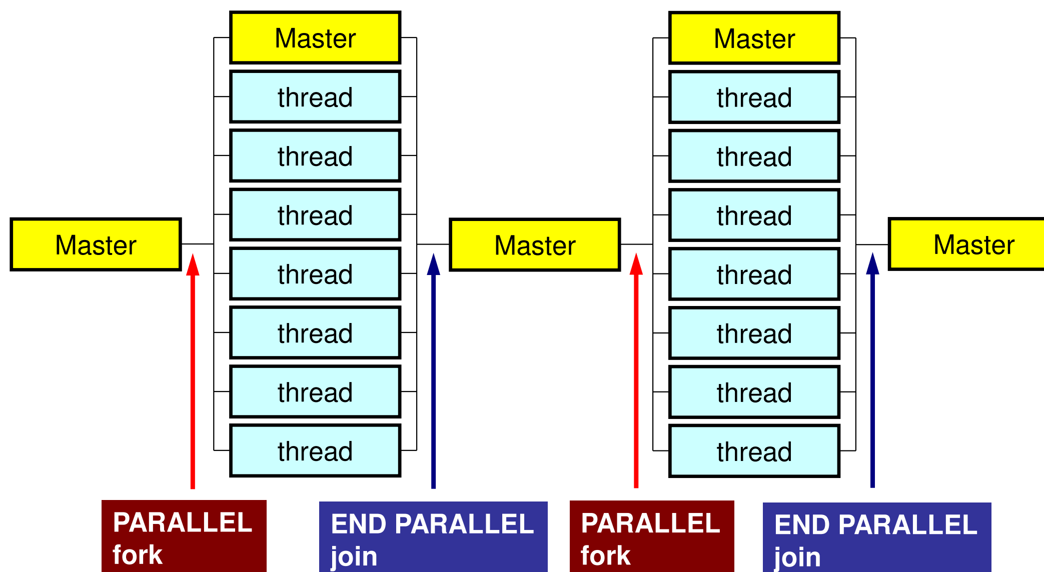
```
 for(i=0; i<N; i++) {
     VAL = D[i] * W[P][i];
     for(j=indexLU[i]; j<indexLU[i+1]; j++) {
         VAL += AMAT[j] * W[P][itemLU[j]];
     }
     W[Q][i] = VAL;
 }
```



```
#pragma omp parallel for private (i,ip.VAL,j)
 for (ip=0; i<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はそのたびにスレッドを生成，消滅させる：fork-join
- ループが連続するとオーバーヘッドになる。
- 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**
- OpenMPによる並列化（0）（12コアまで）
- OpenMPによる並列化（1）（First Touch）
- OpenMPによる並列化（2）（＋ELL）
- OpenMPによる並列化（3）（＋omp-parallel削減）
- OpenMPによる並列化（4）（＋更なる最適化（Fortranのみ））

- OpenMP
- Login to Wisteria/BDEC-01
- **OpenMPによる並列化（0）（12コアまで）**
- OpenMPによる並列化（1）（First Touch）
- OpenMPによる並列化（2）（＋ELL）
- OpenMPによる並列化（3）（＋omp-parallel削減）
- OpenMPによる並列化（4）（＋更なる最適化（Fortranのみ））

ここでの目標

- プログラムの並列化
- OpenMP指示行を入れるだけでやってみる
  - “poi_gen.f/c”(poi_gen),
  - “solver_PCG.f/c” (solve_PCG)

前処理付共役勾配法

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
```

実際にやるべき計算は:

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

「近似逆行列」の計算が必要:

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

究極の前処理:本当の逆行列

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

対角スケーリング:簡単=弱い

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

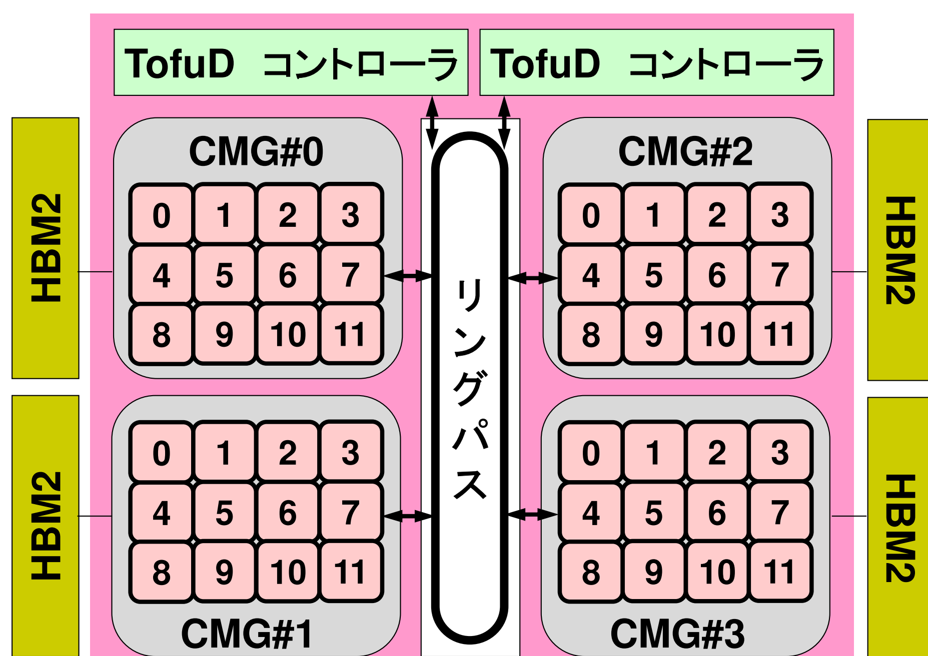
対角スケーリング，点ヤコビ前処理

- 前処理行列として，もとの行列の対角成分のみを取り出した行列を前処理行列 [M] とする。
  - 対角スケーリング，点ヤコビ（point-Jacobi）前処理

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

- **solve [M]z$^{(i-1)}$= r$^{(i-1)}$**という場合に逆行列を簡単に求めることができる。
- 簡単な問題では収束する。

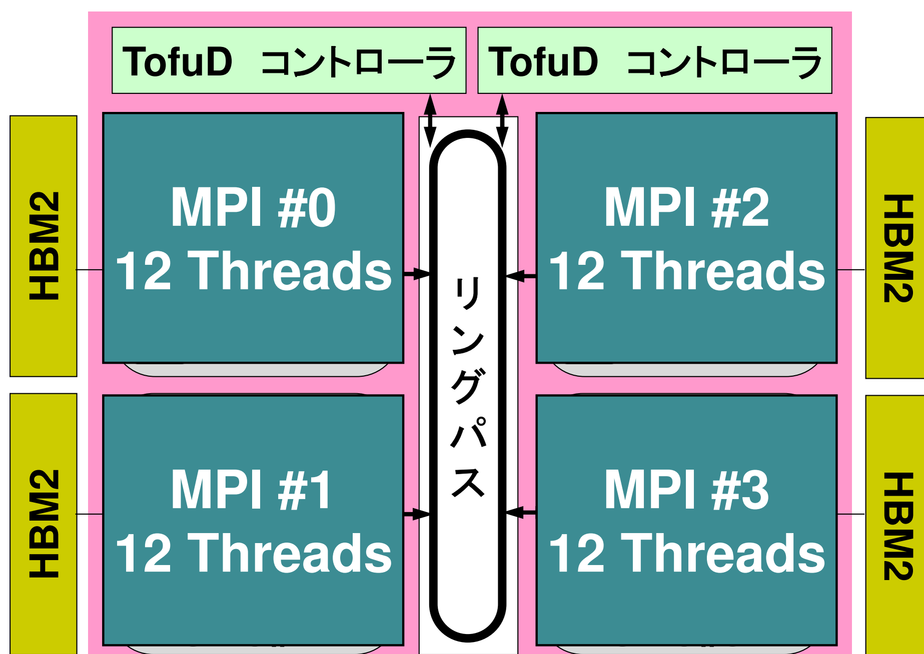
A64FXプロセッサ



プロセッサ名	A64FX
プロセッサ数(コア数)	1 (48+アシスタントコア2 or 4)
周波数	2.2 GHz
理論演算性能	3.3792 TFLOPS
メモリ容量	32 GiB
メモリ帯域幅	1,024 GB/s
L1 Cache	64 KiB/core (Inst/Data)
L2 Cache	8 MiB/CMG

- 4つのCMG（Core Memory Group），12計算コア/CMG
- NUMAアーキテクチャ（Non-Uniform Memory Access）
  - ✓ メモリは各CMGに搭載されていて独立，異なるCMGのローカルメモリ上のデータをアクセスすることは可能
  - ✓ ローカルメモリ上のデータを使って計算するのが効率的
- 大規模並列：各CMGに1-MPIプロセス（12-OpenMPスレッド），プロセッサ内4プロセスのハイブリッド推奨

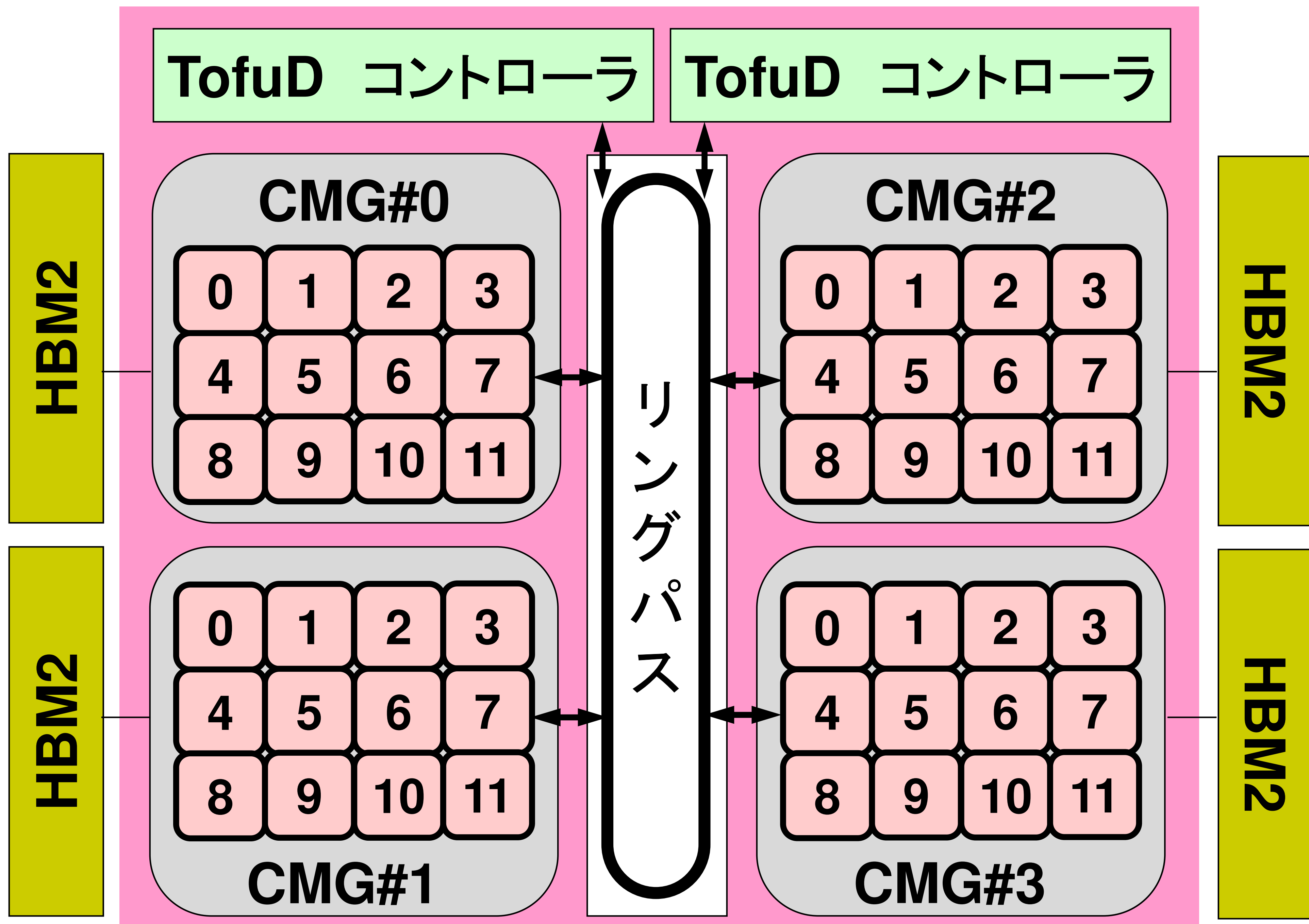
A64FXプロセッサ



プロセッサ名	A64FX
プロセッサ数(コア数)	1 (48+アシスタントコア2 or 4)
周波数	2.2 GHz
理論演算性能	3.3792 TFLOPS
メモリ容量	32 GiB
メモリ帯域幅	1,024 GB/s
L1 Cache	64 KiB/core (Inst/Data)
L2 Cache	8 MiB/CMG

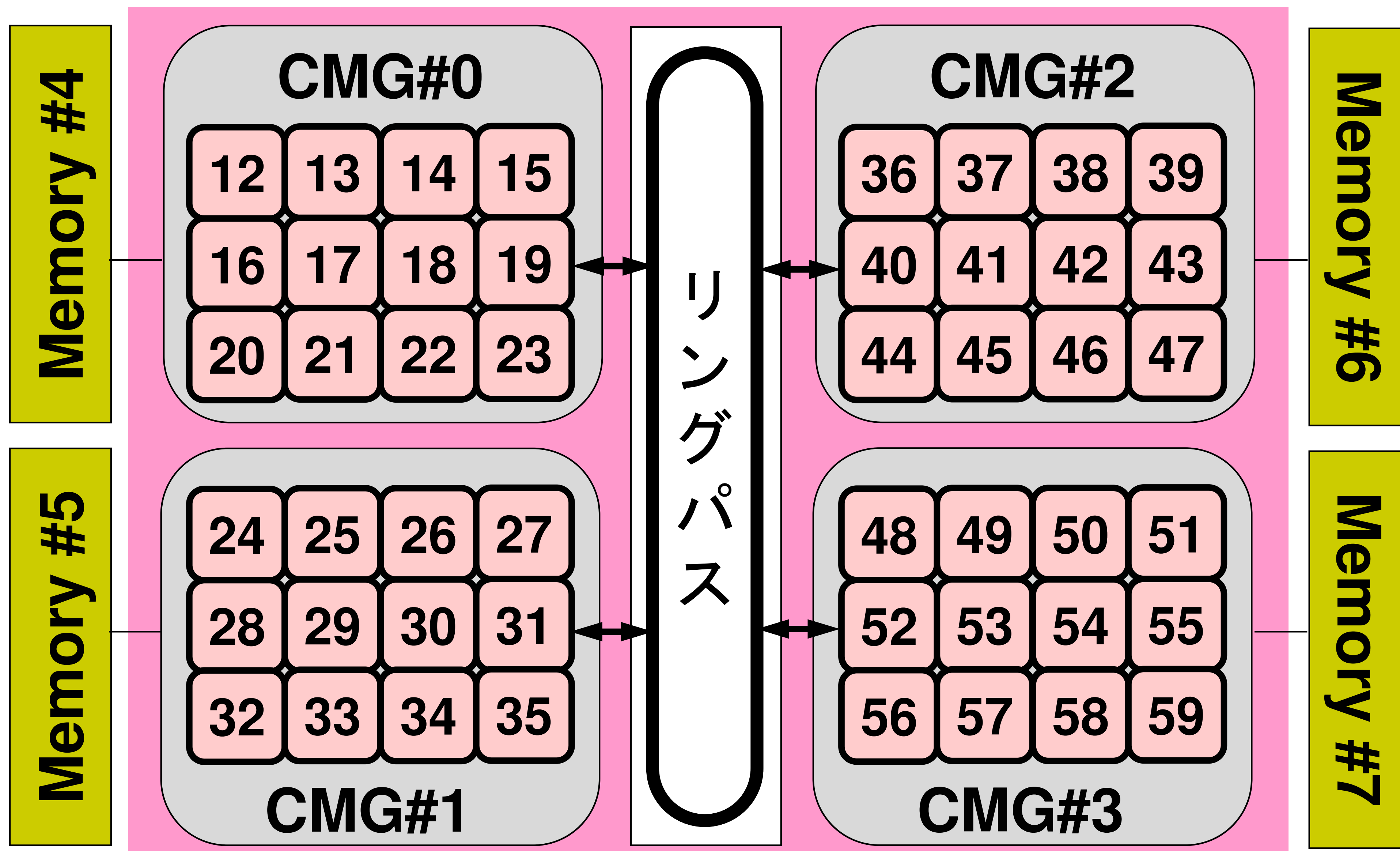
- 4つのCMG（Core Memory Group），12計算コア/CMG
- NUMAアーキテクチャ（Non-Uniform Memory Access）
  - ✓ メモリは各CMGに搭載されていて独立，異なるCMGのローカルメモリ上のデータをアクセスすることは可能
  - ✓ ローカルメモリ上のデータを使って計算するのが効率的
- 大規模並列：各CMGに1-MPIプロセス（12-OpenMPスレッド），プロセッサ内4プロセスのハイブリッド推奨

A64FX：CMG（Core Memory Group）



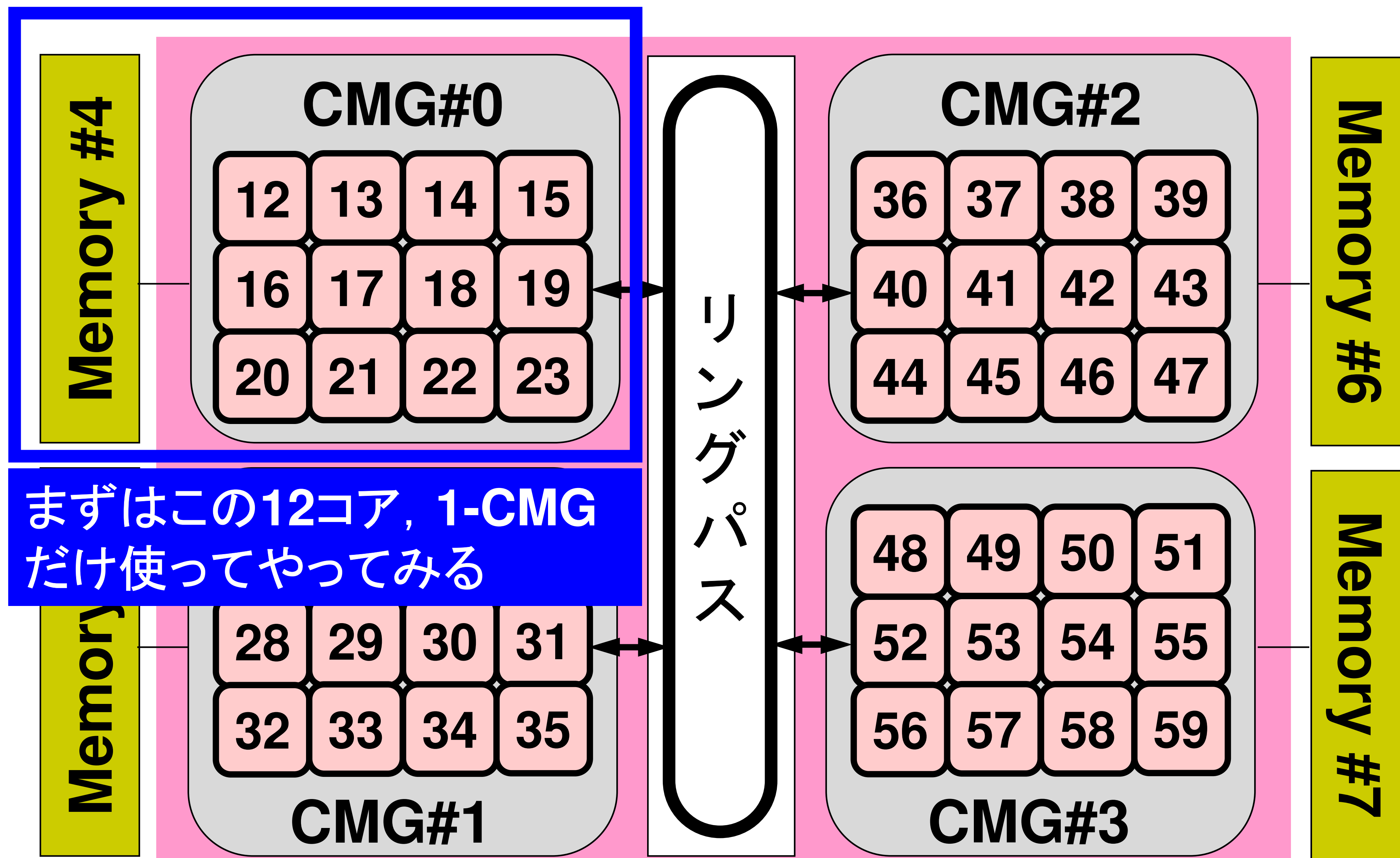
CMG番号，コア番号，メモリ番号（1/2）

CMG:#0-#3，Core:#12-59，Memory:#4-#7



CMG番号，コア番号，メモリ番号（1/2）

CMG:#0-#3，Core:#12-59，Memory:#4-#7



ファイルコピー：Wisteria/BDEC-01

```
>$ cd /work/gt00/t00XYZ
>$ cp /work/gt00/z30088/ompw.tar .

>$ tar xvf ompw.tar

>$ cd ompw                これを<$O-ompw>と呼ぶ
```

以下のディレクトリが出来ていることを確認

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

>$ module load fj  ログインしたら必ずこれをタイプ（コンパイラ）
>$ 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/solc0

.SUFFIXES:
.SUFFIXES: .o .c

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

Cコンパイラ：2種類のモード

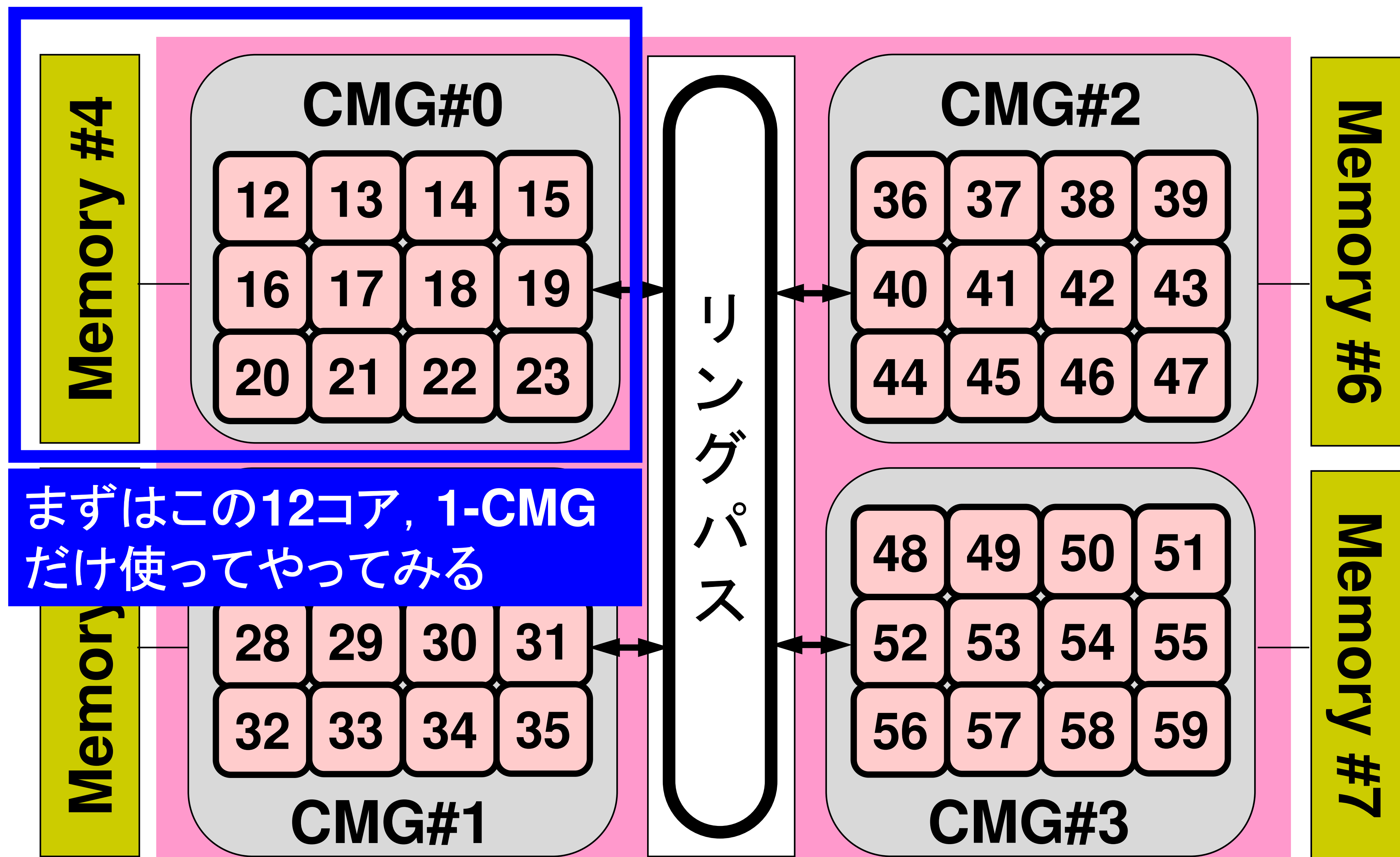
tradモード (-Nnoclang オプション) （デフォルト）	• 京およびPRIMEHPC FX100以前のシステム向け富士通コンパイラをベースとする。 • tradモードは，従来の富士通コンパイラとの互換を重視する場合に適している。 • サポートしている仕様は，C89/C99/C11，OpenMP 3.1/OpenMP 4.5(一部) • オプション省略時（デフォルト）は，-Nnoclangオプション適用 • 本ケースの場合遅い
clangモード (-Nclang オプション)	• オープンソースソフトウェアであるClang/LLVMコンパイラをベースとする。 • clangモードは，最新言語仕様を使用したプログラムや，オープンソースソフトウェアを翻訳する場合に適している。 • サポートしている仕様は，C89/C99/C11， OpenMP 4.5/OpenMP 5.0(一部) • 本ケースの場合，tradモードより速い，最適化すると差は縮まるが，今回はデフォルトでclangモード使用

ジョブ実行

- 実行方法
  - 基本的にバッチジョブのみ
  - インタラクティヴの実行は「基本的に」できません
- 実行手順
  - ジョブスクリプトを書きます
  - ジョブを投入します
  - ジョブの状態を確認します
  - 結果を確認します
- その他
  - 実行時には1ノード（48コア）が占有されます
  - 他のユーザーのジョブに使われることはありません

CMG番号，コア番号，メモリ番号（1/2）

CMG:#0-#3，Core:#12-59，Memory:#4-#7



ジョブスクリプト:c12.sh

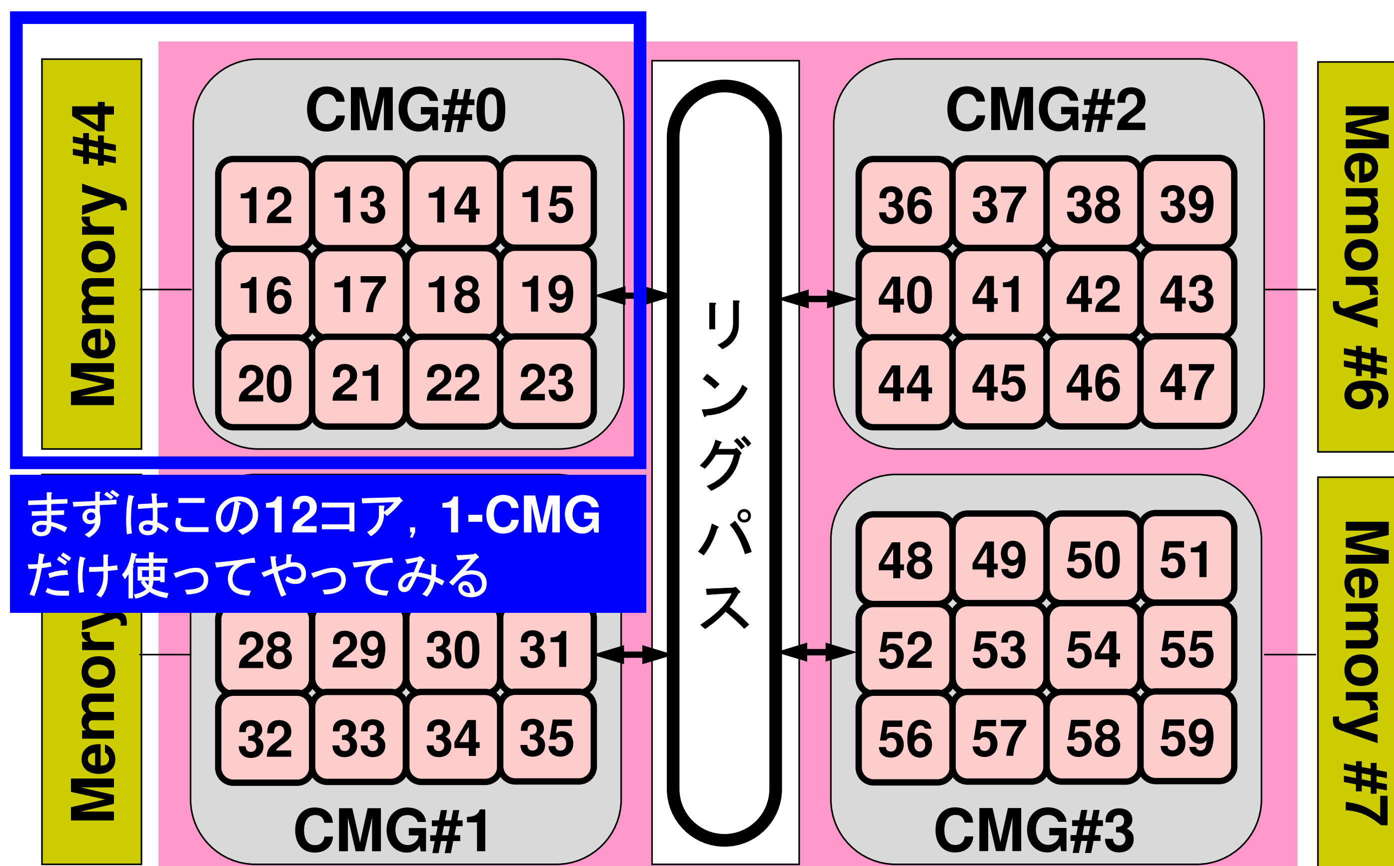
- /work/gt00/t00XXX/ompw/run/c12.sh
- スケジューラへの指令 + シェルスクリプト

```
#!/bin/sh
#PJM -N "c12"                      ジョブ名称（省略可）
#PJM -L rscgrp=tutorial-o          実行キュー名（Resource Group）
#PJM -L node=1                     ノード数（原則＝1）
#PJM --omp thread=12               スレッド数（1-48，しばらくは1-12）
#PJM -L elapse=00:15:00            実行時間
#PJM -g gt00                       グループ名（財布）
#PJM -j
#PJM -e err                        エラー出力ファイル
#PJM -o c12.lst                    標準出力ファイル

module load fj
export OMP_NUM_THREADS=12          スレッド数（--omp thread=XXと同じ数）
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

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

ジョブスクリプト：c12.sh

- `/work/gt00/t00XXX/ompw/run/c12.sh`
- `numactl`：ローカルなリソースを使う

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



ジョブスクリプト：c12.sh

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

環境変数名	指定値（__はdefault）	説明
XOS_MMM_L_PAGING _POLICY	[demand | prepage] [demand | prepage] [demand | prepage]	各メモリ領域のページング方式（ページの割り当て契機）を選択する設定 ✓ demand：デマンドページング方式 ✓ prepage：プリページング方式 本変数はコロン区切りで 3 つのメモリ領域のページング方式を指定： ✓ 第1指定：静的データの .bss 領域，静的データの .data 領域はページング方式指定の対象外で常に prepage ✓ 第2指定：スタック領域・スレッドスタック領域 ✓ 第3 指定は動的メモリ確保領域 指定値以外の値を指定した場合は，「prepage:demand:prepage 」指定とみなす。 **複数CMGをまたぐ場合はdemandを推奨！**

ジョブ投入

```
>$ cd /work/gt00/t00XYZ
>$ 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                              EPSICCG
```

出力(1/2)

```
[t00XYZ@wisteria01 run]$ cat c12.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
    8.634058e+00 sec. (solver)
   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
    8.631781e+00 sec. (solver)
   2097152    1.459831e+04
```

出力(2/2)
5回ずつ実施：ほとんど同じ

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

    5.040907e+00 sec. (solver)
    5.040407e+00 sec. (solver)
    5.038814e+00 sec. (solver)
    5.042296e+00 sec. (solver)
    5.033407e+00 sec. (solver)

    5.034004e+00 sec. (solver)
    5.035210e+00 sec. (solver)
    5.047046e+00 sec. (solver)
    5.043275e+00 sec. (solver)
    5.036129e+00 sec. (solver)
```

```
numactl ./solc0
```

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

利用可能なResource Group（Queue）

- 以下の2種類のResource Groupを利用可能
- 最大12ノードを使える
  - **`lecture-o`**
    - 12ノード（576コア），15分，アカウント有効期間中利用可能
    - 全教育ユーザーで共有
  - **`tutorial-o`**
    - 12ノード（576コア），15分，講義・演習実施時間帯
    - **`Lecture-o`**よりは多くのジョブを投入可能（混み具合による）

様々なコマンド

- ジョブ実行 `pjsub SCRIPT_NAME`
- ジョブ実行状況 `pjstat`
- ジョブ停止 `pjdel JOB_ID`
- ジョブキューの状況 `pjstat --rsc`
- ジョブキューの状況(詳細) `pjstat --rsc -x`
- 実行ジョブ情報 `pjstat -a`
- ジョブ実行履歴 `pjstat -H`
- ジョブ実行制限 `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
tutorial-o               [DISABLE,STOP]        2x12x16

SYSTEM: Aquarius
RSCGRP                   STATUS                   NODE  GPU
lecture-a                [ENABLE,START]              7   56
tutorial-a               [DISABLE,STOP]              2   16

[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
tutorial-o               [DISABLE,STOP]         1       12   00:15:00      --:--:--       28 gt00

SYSTEM: Aquarius
RSCGRP                   STATUS          MIN_NODE MAX_NODE AVAIL_GPU MAX_ELAPSE REMAIN_ELAPSE MEM(GiB) PROJECT
lecture-a                [ENABLE,START]         1        1     1,2,4   00:15:00      00:15:00      448 gt00
tutorial-a               [DISABLE,STOP]         1        1     1,2,4   00:15:00      --:--:--      448 gt00

[t00XYZ@wisteria01 ~]$ pjstat --limit
SYSTEM: Odyssey
PROJECT          ACCEPT           RUN  BULK_ACCEPT    BULK_RUN           NODE
gt00           0/   128      0/    16       0/     8      0/    16       0/ 2304

SYSTEM: Aquarius
PROJECT          ACCEPT           RUN  BULK_ACCEPT    BULK_RUN            GPU
gt00           0/     4      0/     2       0/     0      0/     0       0/   64
```

poi_gen (1/2)：主要部

とにかくprivateに注意

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

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

                        VOL0 = VOLCEL[icel];
                        isLU=indexLU[icel];
                        icou= 0;
                        if(icN5 != 0) {
                                coef = RDZ * ZAREA;
                                D[icel] -= coef;
                                itemLU[icou+isLU]= icN5-1;
                                AMAT  [icou+isLU]= coef;
                                icou= icou + 1;
                        }
(…)
                        ii = XYZ[icel][0];
                        jj = XYZ[icel][1];
                        kk = XYZ[icel][2];

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

poi_gen (2/2)：境界条件部

とにかくprivateに注意

```
#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
若干インタフェース変更
restrict：ポインタが他の変数から参照されない

```
#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 $i= 1, 2, \ldots$
- 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();

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];
        }
}
```

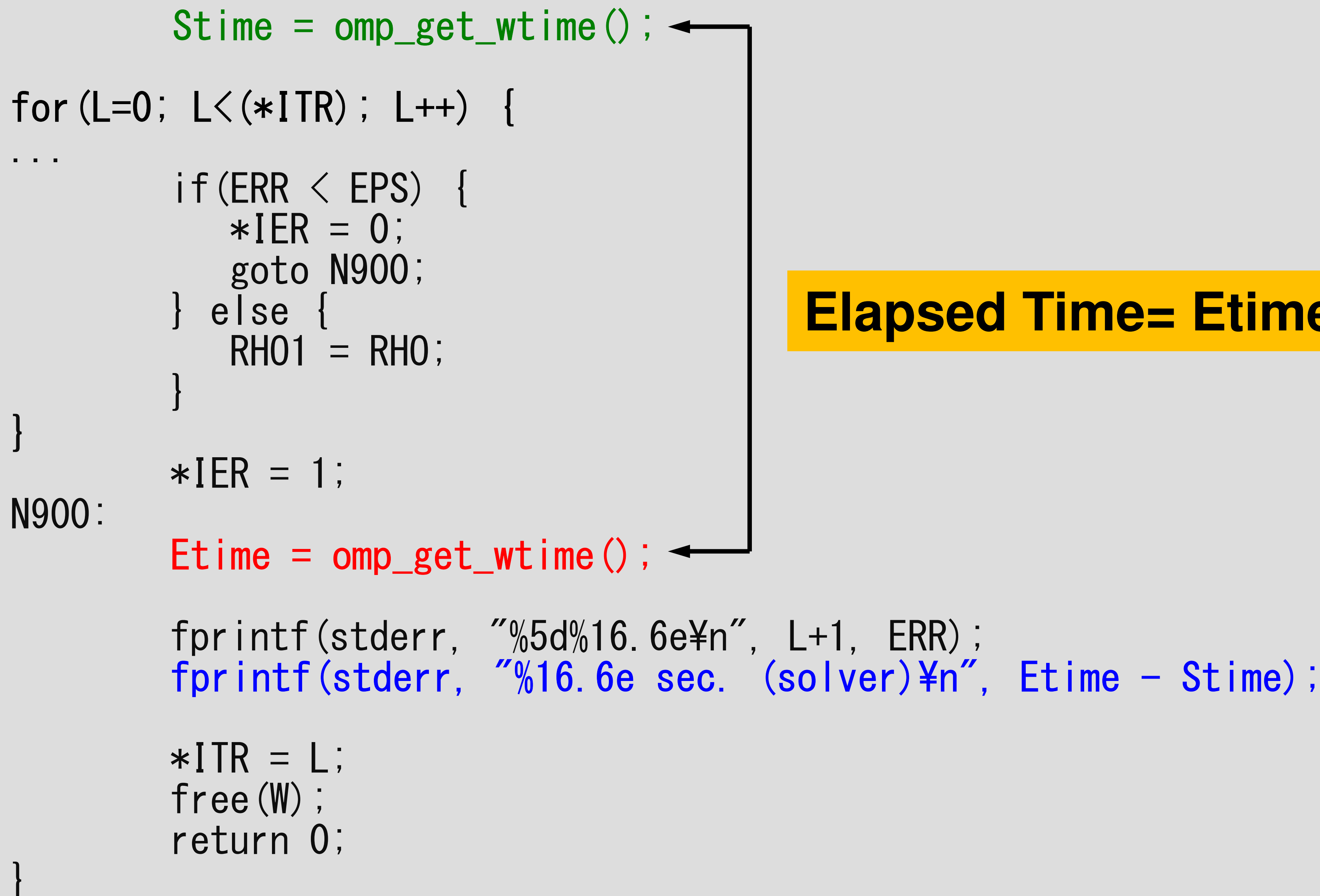
Compute $r^{(0)}= b-[A]x^{(0)}$
<u>for</u> i= 1, 2, …
    solve $[M]z^{(i-1)}= r^{(i-1)}$
    $\rho_{i-1}= r^{(i-1)} z^{(i-1)}$
    <u>if</u> i=1
      $p^{(1)}= z^{(0)}$
    <u>else</u>
      $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
      $p^{(i)}= z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
    <u>endif</u>
  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

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

solve_PCG (5/5)



```
        Stime = omp_get_wtime();

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

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

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

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

- /work/gt00/t00XXX/ompw/run/cYZ.sh
- スケジューラへの指令 + シェルスクリプト

```
#!/bin/sh
#PJM -N "cYZ"                     ジョブ名称（省略可）
#PJM -L rscgrp=tutorial-o         実行キュー名（Resource Group）
#PJM -L node=1                    ノード数（原則＝1）
#PJM --omp thread=YZ              スレッド数（1-48, しばらくは1-12）
#PJM -L elapse=00:15:00           実行時間
#PJM -g gt00                      グループ名（財布）
#PJM -j
#PJM -e err                       エラー出力ファイル
#PJM -o cYZ.lst                   標準出力ファイル

module load fj
export OMP_NUM_THREADS=YZ         スレッド数（--omp thread=YZと同じ数）
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

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

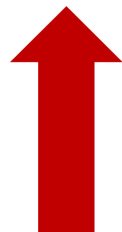
PCG計算時間: Etime-Stime：Fortran

NX=NY=NZ=128

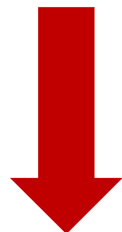
実行時間:5回測定して最速時間採用

コア数増加による効率低下:メモリ利用効率の低下

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=tutorial-o
#PJM   -L     node=1
#PJM   --omp  thread=4
#PJM   -L     elapse=00:15:00
#PJM   -g     gt00
#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       -l ./solc0
numactl       -l ./solc0
numactl       -l ./solc0
numactl       -l ./solc0
numactl       -l ./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

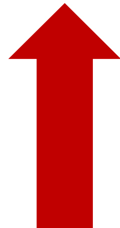
```
#!/bin/sh
#PJM   -N     "c08"
#PJM   -L     rscgrp=tutorial-o
#PJM   -L     node=1
#PJM   --omp  thread=8
#PJM   -L     elapse=00:15:00
#PJM   -g     gt00
#PJM   -j
#PJM   -e     err
#PJM   -o     c08.lst

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

numactl       -l ./solc0
numactl       -l ./solc0
numactl       -l ./solc0
numactl       -l ./solc0
numactl       -l ./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
```

CMG数増加
PCG計算時間: Etime-Stime：Fortran

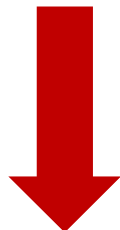
NX=NY=NZ=128

実行時間:5回測定して最速時間採用

Large
大



Granularity：粒度
Problem Size/Thread



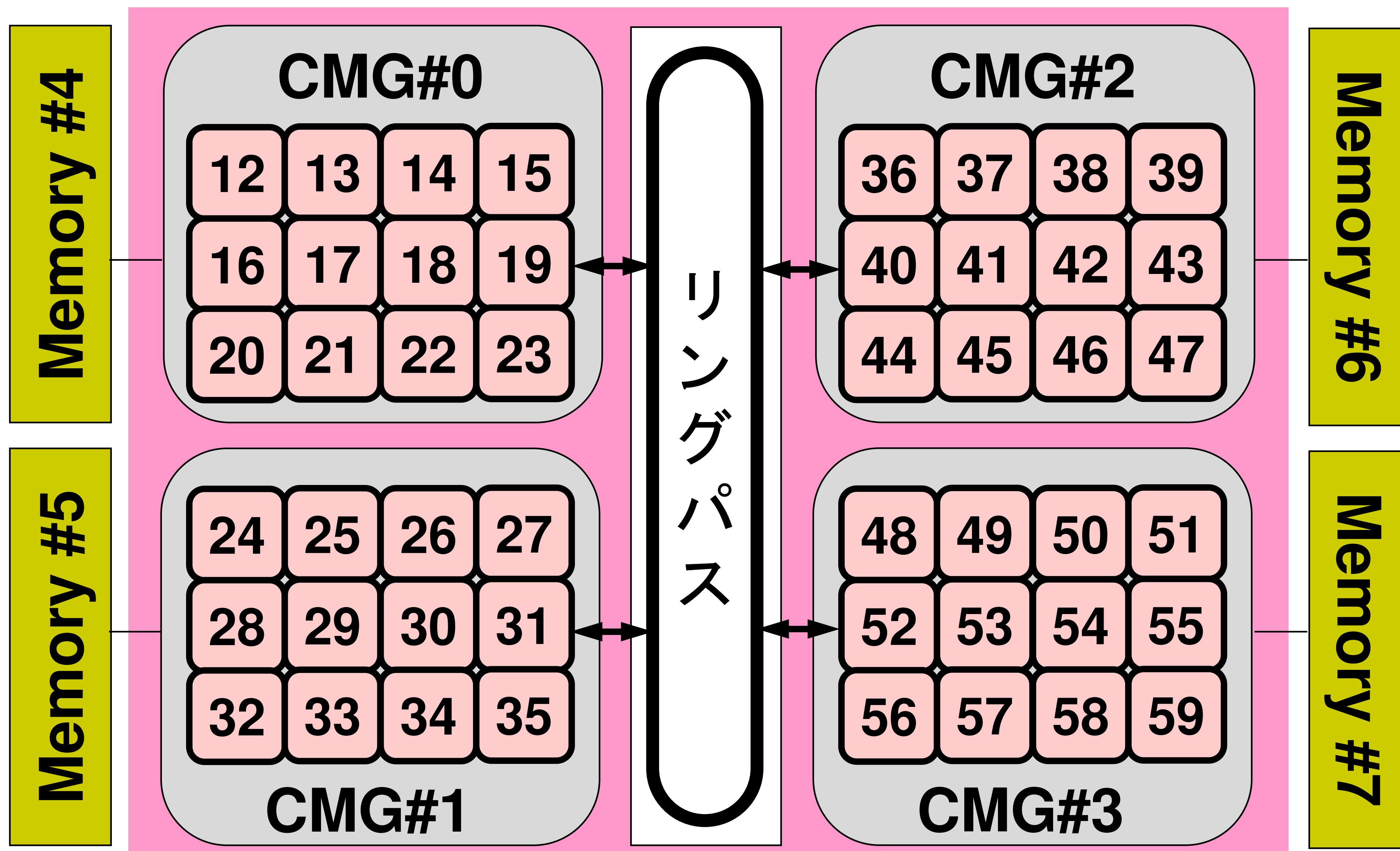
Small
小

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#

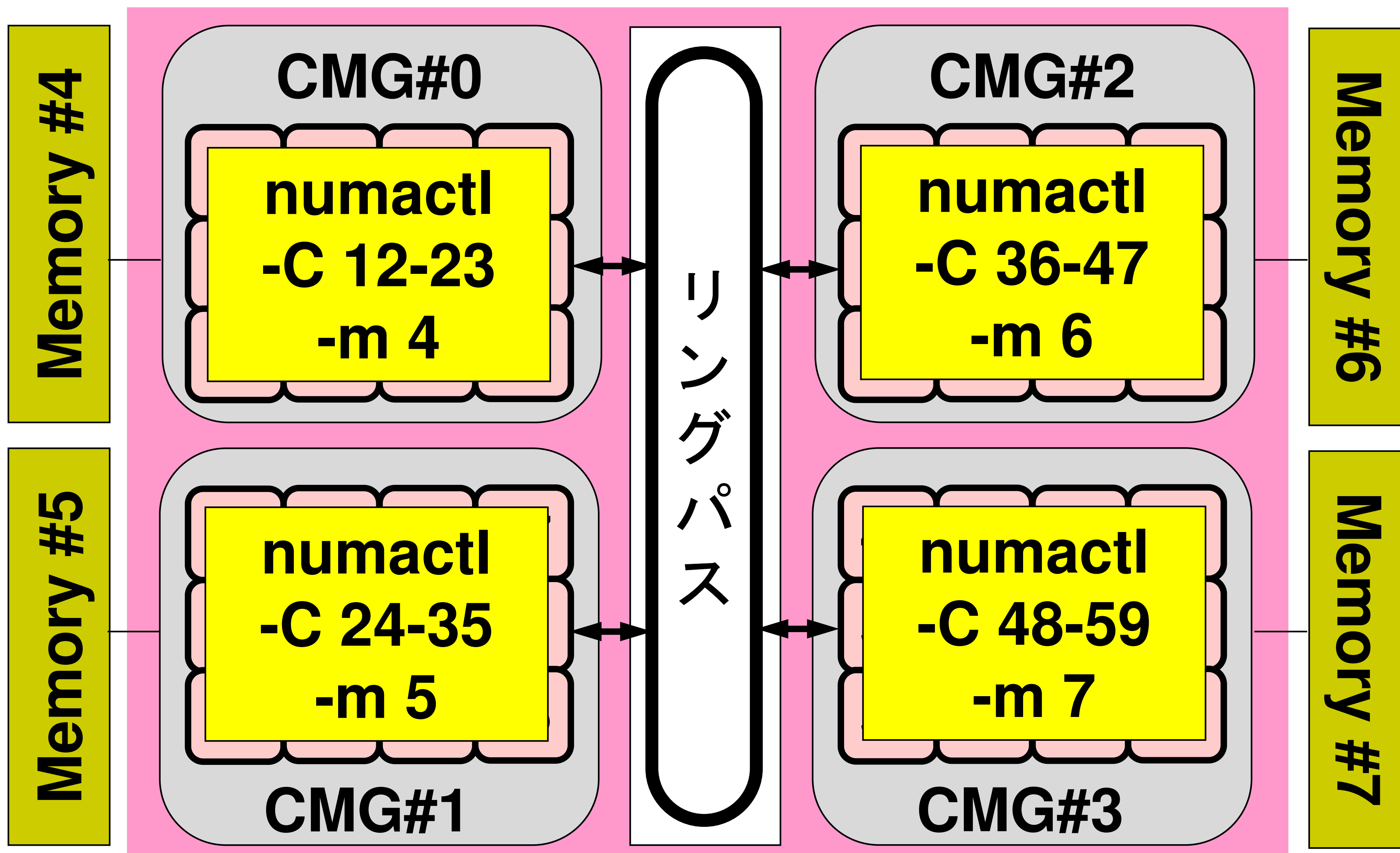
CMG番号，コア番号，メモリ番号（1/2）

CMG:#0-#3，Core:#12-59，Memory:#4-#7



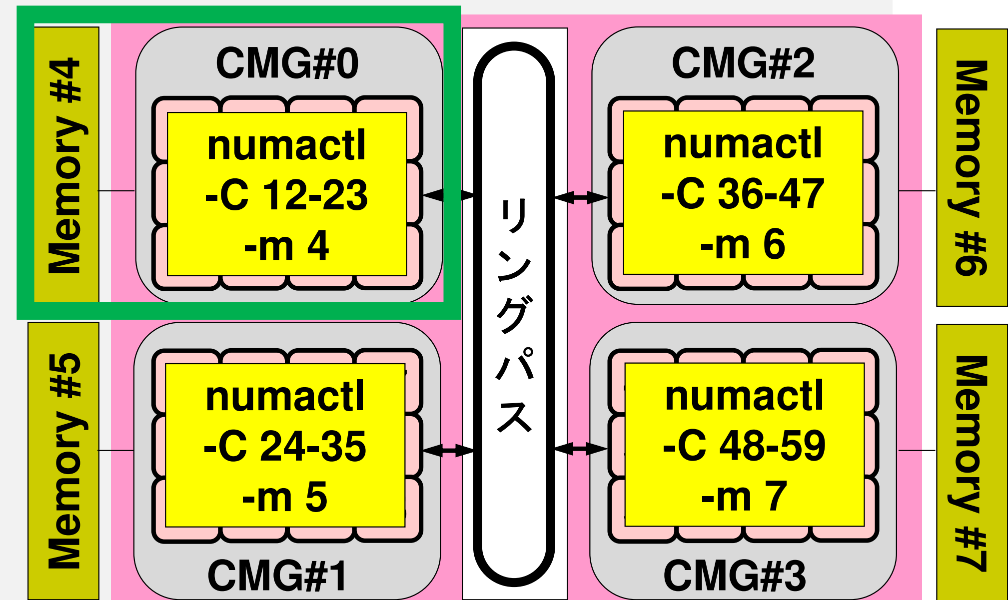
CMG番号，コア番号，メモリ番号（2/2）

CMG:#0-#3，Core:#12-59，Memory:#4-#7



c0_12.sh

```
#!/bin/sh
#PJM   -N     "c0_12"
#PJM   -L     rscgrp=tutorial-o
#PJM   -L     node=1
#PJM   --omp  thread=12
#PJM   -L     elapse=00:15:00
#PJM   -g     gt00
#PJM   -j
#PJM   -e     err
#PJM   -o     c0_12.lst

module load   fj
export OMP_NUM_THREADS=12
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl       -l ./solc0
numactl       -l ./solc0
numactl       -l ./solc0
numactl       -l ./solc0
numactl       -l ./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
```

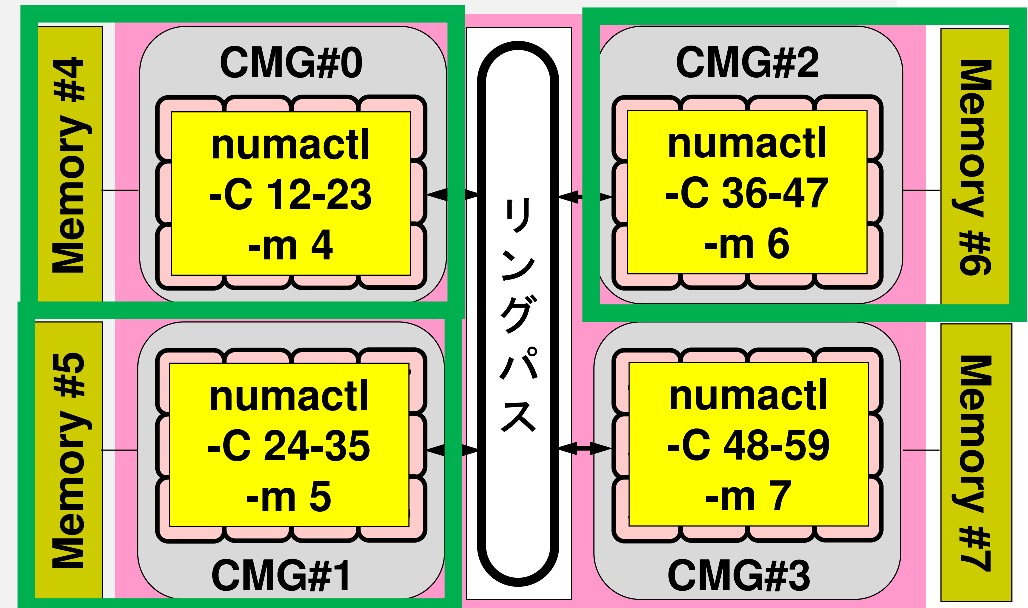


c0_24.sh

```
#!/bin/sh
#PJM   -N     "c0_24"
#PJM   -L     rscgrp=tutorial-o
#PJM   -L     node=1
#PJM   --omp  thread=24
#PJM   -L     elapse=00:15:00
#PJM   -g     gt00
#PJM   -j
#PJM   -e     err
#PJM   -o     c0_24.lst

module load   fj
export OMP_NUM_THREADS=24
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl       -l ./solc0
numactl       -l ./solc0
numactl       -l ./solc0
numactl       -l ./solc0
numactl       -l ./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
```

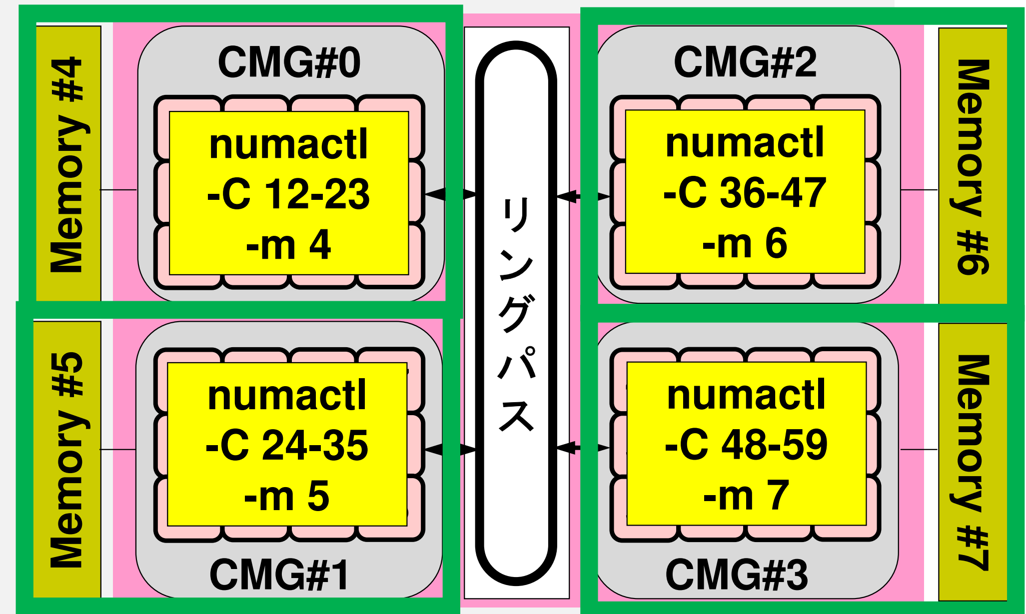
c0_36.sh

```
#!/bin/sh
#PJM    -N      "c0_36"
#PJM    -L      rscgrp=tutorial-o
#PJM    -L      node=1
#PJM    --omp   thread=36
#PJM    -L      elapse=00:15:00
#PJM    -g      gt00
#PJM    -j
#PJM    -e      err
#PJM    -o      c0_36.lst

module load   fj
export OMP_NUM_THREADS=36
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl         -l ./solc0
numactl         -l ./solc0
numactl         -l ./solc0
numactl         -l ./solc0
numactl         -l ./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
```



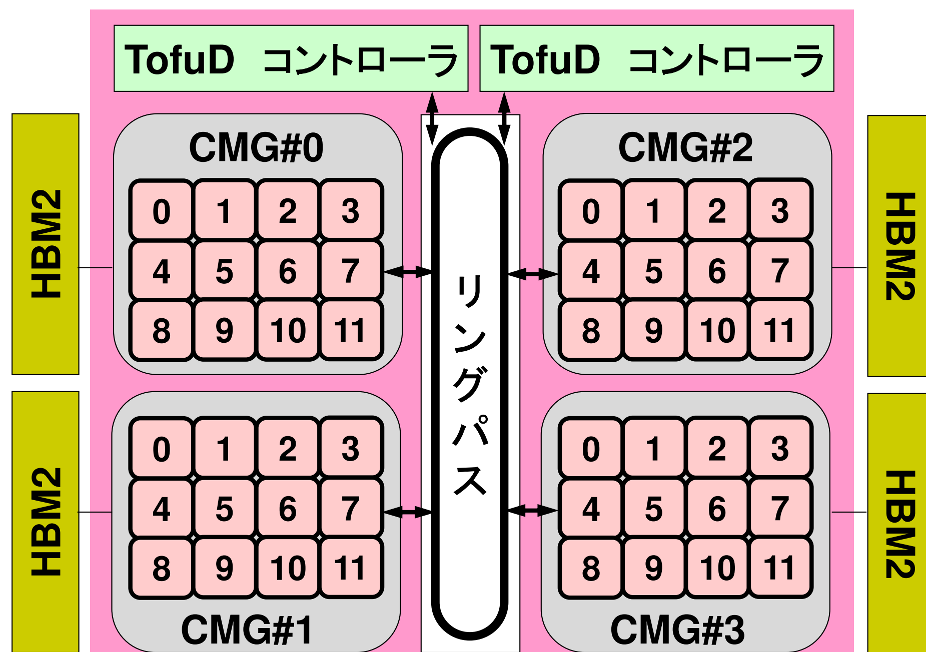
c0_48.sh

```
#!/bin/sh
#PJM    -N      "c0_48"
#PJM    -L      rscgrp=tutorial-o
#PJM    -L      node=1
#PJM    --omp   thread=48
#PJM    -L      elapse=00:15:00
#PJM    -g      gt00
#PJM    -j
#PJM    -e      err
#PJM    -o      c0_48.lst

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

numactl        -l ./solc0
numactl        -l ./solc0
numactl        -l ./solc0
numactl        -l ./solc0
numactl        -l ./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
- OpenMPによる並列化（0）（12コアまで）
- **OpenMPによる並列化（1）（First Touch）**
- OpenMPによる並列化（2）（＋ELL）
- OpenMPによる並列化（3）（＋omp-parallel削減）
- OpenMPによる並列化（4）（＋更なる最適化（Fortranのみ））

A64FXプロセッサ



プロセッサ名	A64FX
プロセッサ数(コア数)	1 (48+アシスタントコア2 or 4)
周波数	2.2 GHz
理論演算性能	3.3792 TFLOPS
メモリ容量	32 GB
メモリ帯域幅	1,024 GB/s

- 4つのCMG（Core Memory Group），12計算コア/CMG
- NUMAアーキテクチャ（Non-Uniform Memory Access）
  - ✓ メモリは各CMGに搭載されていて独立，異なるCMGのローカルメモリ上のデータをアクセスすることは可能
  - ✓ ローカルメモリ上のデータを使って計算するのが効率的
- 大規模並列：各CMGに1-MPIプロセス（12-OpenMPスレッド），プロセッサ内4プロセスのハイブリッド推奨

データをローカルメモリに置く方法（**1/2**）

- NUMAアーキテクチャでは，プログラムにおいて変数や配列を宣言した時点では，物理的メモリ上に記憶領域は確保されず，ある変数を最初にアクセスしたコア（の属するCMG）のローカルメモリ上に，その変数の記憶領域（ページ）が確保される。
- これを**First Touch Data Placement**（**First Touch**）と呼び，配列の初期化手順により得られる性能が大幅に変化する場合があるため，注意が必要である。
- 例えばある配列を初期化する場合，特に指定しなければ0番のCMGで初期化が行われるため，記憶領域は0番CMGのローカルメモリ上に確保される。

データをローカルメモリに置く方法（2/2）

- したがって，他のCMGでこの配列のデータをアクセスする場合には，必ず0番CMGのメモリにアクセスする必要があるため，高い性能を得ることは困難である。
- 配列の初期化を，実際の計算の手順にしたがってOpenMPを使って並列に実施すれば，実際に計算を担当するCMGのメモリにその配列の担当部分の記憶領域が確保され，より効率的に計算を実施することができる。
- 1CMGしか使用しない場合はこのような配慮は不要
  - OpenMP/MPIハイブリッドで1-CMG当りに1つのMPIプロセスを使用する場合も同様
  - numactl

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.

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 is to initialize data in parallel using the same loop schedule as will be used later in the computations.

First-Touchに対応したバージョン：src-c1 オリジナルは src-c0

```
>$ cd /work/gt00/t00XYZ/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;
        }
---

#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-c0

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;
          }
        }
```

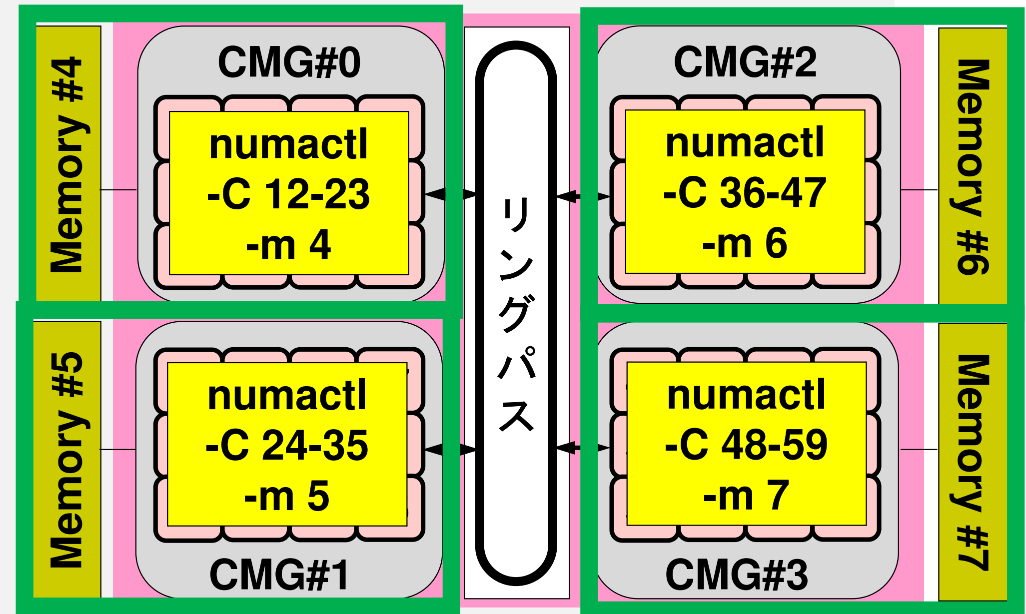
src-c0

```
#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=tutorial-o
#PJM   -L     node=1
#PJM   --omp  thread=48
#PJM   -L     elapse=00:15:00
#PJM   -g     gt00
#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        -l ./solc1
numactl        -l ./solc1
numactl        -l ./solc1
numactl        -l ./solc1
numactl        -l ./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
```



CMG数増加
PCG計算時間: Etime-Stime：Fortran

NX=NY=NZ=128

実行時間：5回測定して最速時間採用

	Thread #	sec	Speed-up	Parallel Efficiency (%)
src-f0	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

CMG数増加
PCG計算時間: Etime-Stime：C（clang）

NX=NY=NZ=128

実行時間：5回測定して最速時間採用

	Thread #	sec	Speed-up	Parallel Efficiency (%)
src-c0	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

CMG数増加
PCG計算時間: Etime-Stime

NX=NY=NZ=128

実行時間:5回測定して最速時間採用

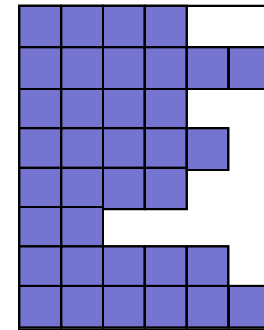
	Language	12	24	36	48
Original	Fortran	5.09	2.79	1.99	1.70
	C (clang)	5.03	2.69	1.91	1.57
	C (trad)	7.75	4.19	2.90	2.36
First-Touch	Fortran	5.27	2.70	1.87	1.52
	C (clang)	5.22	2.70	1.85	1.17
	C (trad)	7.85	4.05	2.79	1.72 (112.6 %)

- OpenMP
- Login to Wisteria/BDEC-01
- OpenMPによる並列化（0）（12コアまで）
- OpenMPによる並列化（1）（First Touch）
- **OpenMPによる並列化（2）（＋ELL）**
- OpenMPによる並列化（3）（＋omp-parallel削減）
- OpenMPによる並列化（4）（＋更なる最適化（Fortranのみ））

疎行列の格納方式

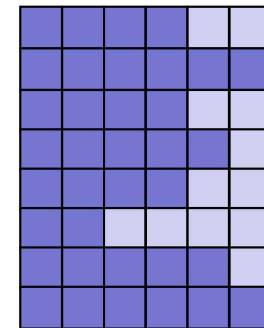
CRS（Compressed Row Storage）

```
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;}
```

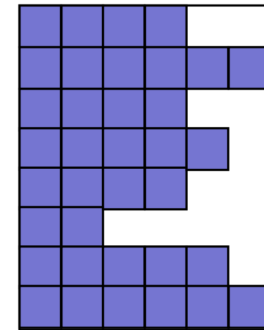


- CRS: Compressed Row Storage
  - 非ゼロ非対角成分のみ格納⇒メモリ節約できるが，計算効率悪い
- ELL：ELLPACK/ITPACK
  - 非ゼロ非対角成分数固定⇒0のところには0を入れる
  - 記憶容量，計算量は増えるがメモリアクセス性能向上⇒Prefetch

疎行列の格納方式

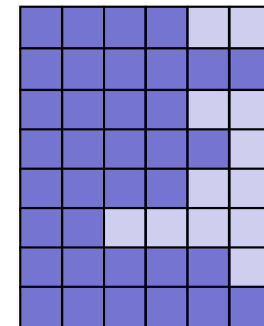
CRS（Compressed Row Storage）

```
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)　こちらの方がだいぶ遅い

```
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に注意

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

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

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

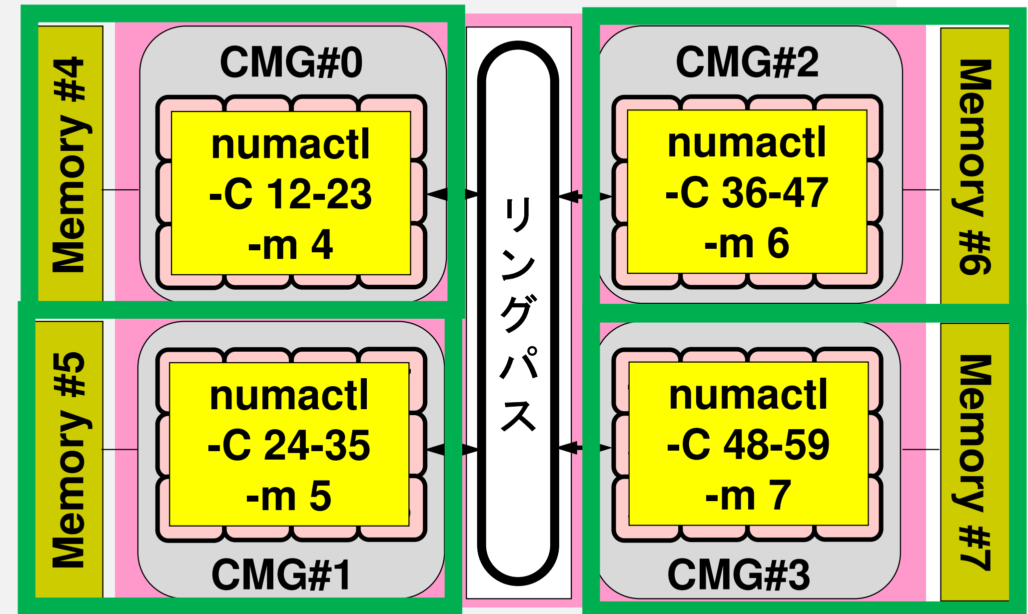
```
/*****************
 * {q} = [A]{p} *
 *****************/

#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;
                }
```

ELLに対応したバージョン src-c2

```
>$ cd /work/gt00/t00XYZ/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=tutorial-o
#PJM   -L      node=1
#PJM   --omp   thread=48
#PJM   -L      elapse=00:15:00
#PJM   -g      gt00
#PJM   -j
#PJM   -e      err
#PJM   -o      c2_48.lst

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

numactl       -l ./solc2
numactl       -l ./solc2
numactl       -l ./solc2
numactl       -l ./solc2
numactl       -l ./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
```



出力(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
```

出力(2/2):Fortran, 5回実施:ほぼ同じ

```
[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)

    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 -l ./solf1

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)

    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 -l ./solf2

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

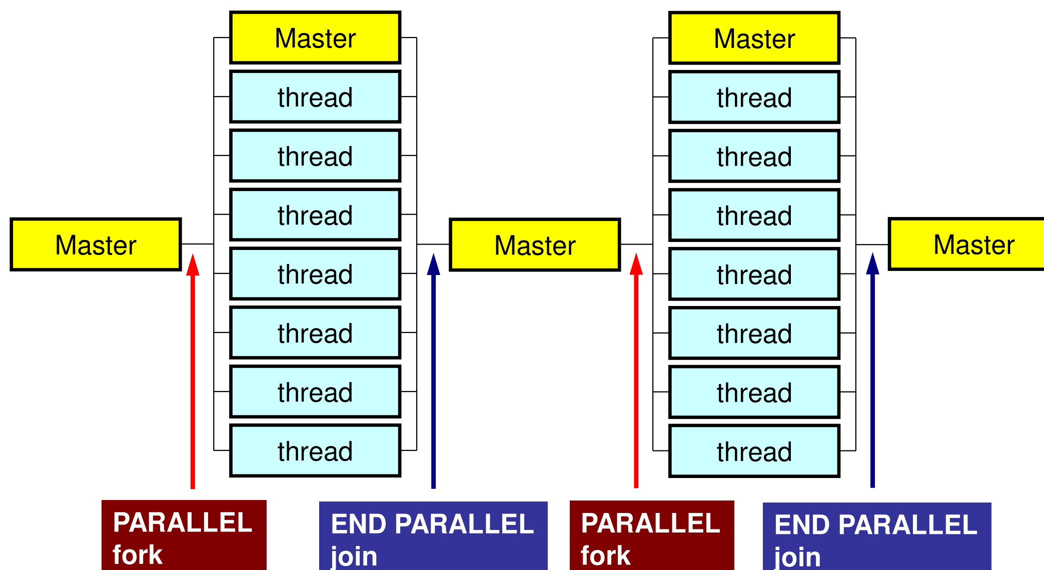
- OpenMP
- Login to Wisteria/BDEC-01
- OpenMPによる並列化（0）（12コアまで）
- OpenMPによる並列化（1）（First Touch）
- OpenMPによる並列化（2）（＋ELL）
- **OpenMPによる並列化（3）（＋omp-parallel削減）**
- **OpenMPによる並列化（4）（＋更なる最適化（Fortranのみ））**

omp parallel (do)

- omp parallel-omp end parallelはそのたびにスレッドを生成，消滅させる：fork-join
- ループが連続するとオーバーヘッドになる。
- 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
```



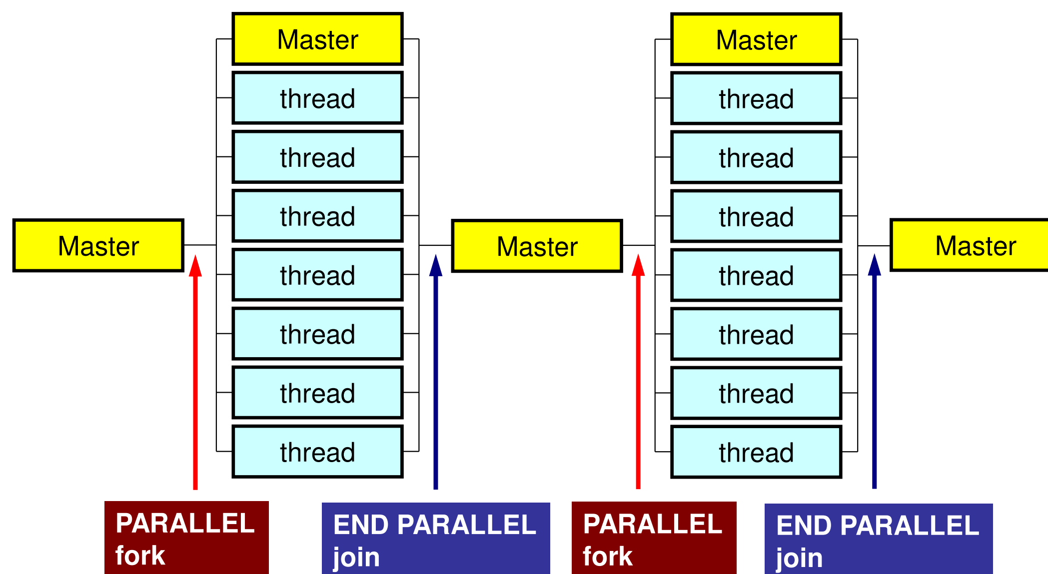
omp parallel for：スレッド生成消滅

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

更なる最適化の方針

- src-c3, src-f3
  - 各反復で一回だけomp parallel を呼ぶ
- src-f4（Fortranのみ）
  - !$omp parallelは一回のみ呼び出す（反復の最初と最後）
  - !$omp doもなくす
  - 内積部分のreductionは並列化しない



更なる最適化の方針：Fortranの場合

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 $i= 1, 2, \ldots$

    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

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);
```

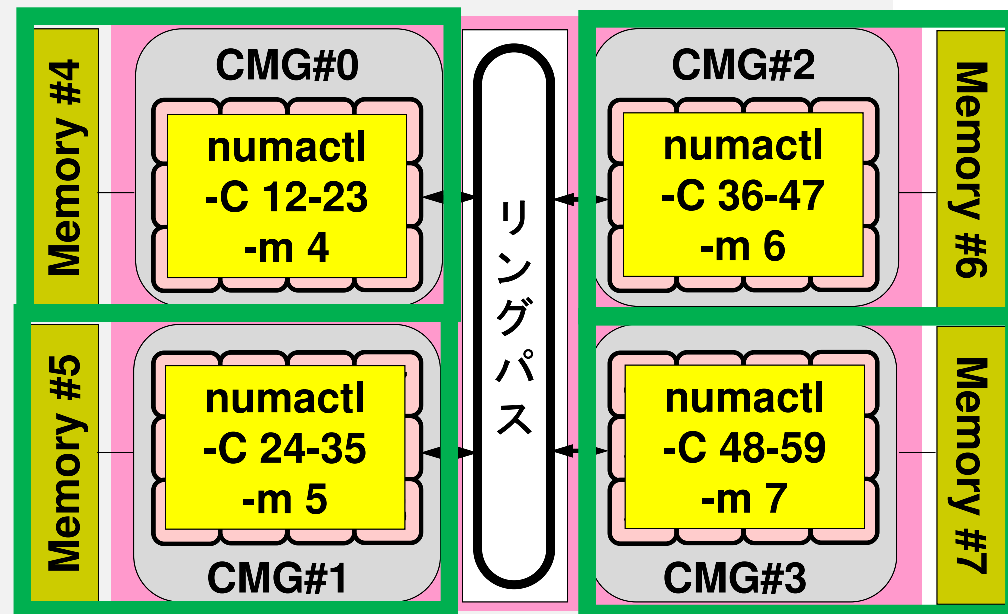
Compute $r^{(0)}= b-[A]x^{(0)}$
<u>for</u> i= 1, 2, …
    solve $[M]z^{(i-1)}= r^{(i-1)}$
    $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
    <u>if</u> i=1
        $p^{(1)}= z^{(0)}$
    <u>else</u>
        $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
        $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
    <u>endif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

src-c3 に対応したバージョン

```
>$ cd /work/gt00/t00XYZ/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=tutorial-o
#PJM   -L     node=1
#PJM   --omp  thread=48
#PJM   -L     elapse=00:15:00
#PJM   -g     gt00
#PJM   -j
#PJM   -e     err
#PJM   -o     c3_48.lst

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

numactl       -l ./solc3
numactl       -l ./solc3
numactl       -l ./solc3
numactl       -l ./solc3
numactl       -l ./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言語：tradを使う場合

```
>$ cd /work/gt00/t00XYZ/ompw
>$ cd run

<modify "INPUT.DAT", "c48org.sh">

>$ pjsub c48org.sh
```

c48org.sh

```
#!/bin/sh
#PJM -N "cx48"
#PJM -L rscgrp=lecture-o
#PJM -L node=1
#PJM --omp thread=48
#PJM -L elapse=00:15:00
#PJM -g gt00
#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 -l ./solc0org
numactl -l ./solc0org
numactl -l ./solc0org
numactl -l ./solc0org
numactl -l ./solc0org
…
```

PCG法：計算時間：N=128^3，48スレッド N= 2,097,152

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

PCG法：計算時間：N=160^3，48スレッド N= 4,096,000

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

PCG法：計算時間：N=200^3，48スレッド N= 8,000,000

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

PCG法：計算時間：N=256^3，48スレッド N= 16,777,216

	Fortran	C (clang)	C (trad)
src0（初期設定）	34.308	24.772	25.547
src1（First Touch）	32.202	22.172	23.814
src2（+ELL）	8.916	10.761	14.566
src3（+omp-parallel削減）	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_RHO, 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

!$omp parallel
      PEsmpTOT= omp_get_num_threads()     PEsmpTOT:総スレッド数取得
!$omp end parallel

      allocate (SMPindex(0:PEsmpTOT))     SMPindex(0:PEsmpTOT):各スレッド受持要素番号
      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

      allocate (W_RHO(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
```

src_f4 (4/5)

```
      ITR= N
      Stime= omp_get_wtime()

!$omp parallel private(L,ip,ip1,ip2,i,k,VAL)
!$omp&          private(RHO,BETA,RHO1,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, PEsmpTOT
        RHO= RHO + W_RHO(i)
      enddo

      if ( L.eq.1 ) then
!$omp simd
        do i= ip1, ip2
          W(i,P)= W(i,Z)
        enddo
       else
        BETA= RHO / RHO1
!$omp simd
        do i= ip1, ip2
          W(i,P)= W(i,Z) + BETA*W(i,P)
        enddo
      endif
!$omp barrier
```

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

src_f4 (5/5)

```
      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= RHO / 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)...
```

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

<u>for</u> i= 1, 2, …

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

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

    <u>if</u> i=1

        $p^{(1)}= z^{(0)}$

     <u>else</u>

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

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

    <u>endif</u>

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

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

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

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

    check convergence |r|

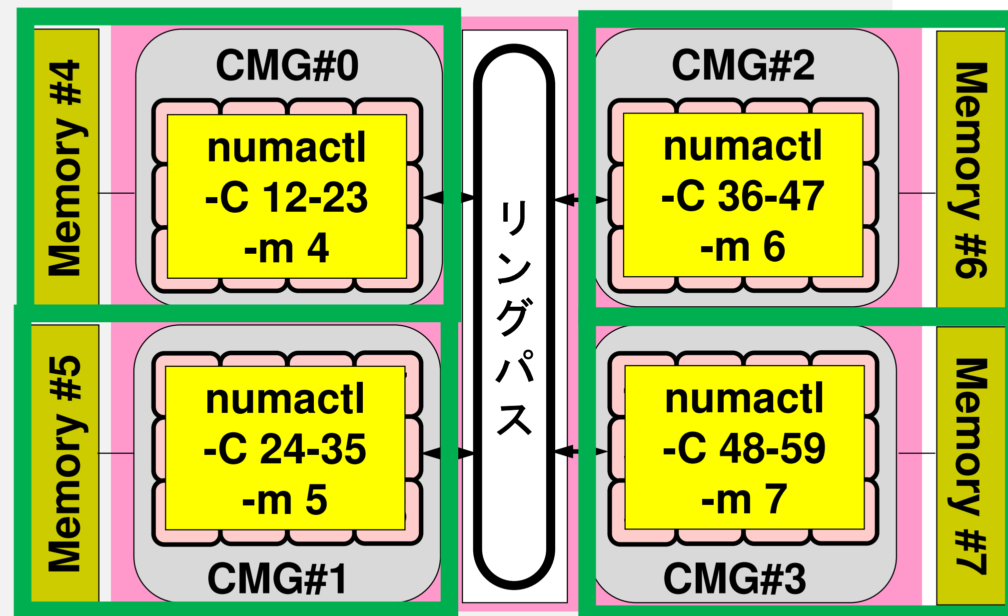
<u>end</u>

src4 に対応したバージョン

```
>$ cd /work/gt00/t00XYZ/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=tutorial-o
#PJM   -L     node=1
#PJM   --omp  thread=48
#PJM   -L     elapse=00:15:00
#PJM   -g     gt00
#PJM   -j
#PJM   -e     err
#PJM   -o     f4_48.lst

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

numactl       -l ./solf4
numactl       -l ./solf4
numactl       -l ./solf4
numactl       -l ./solf4
numactl       -l ./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
```



PCG法：計算時間：N=128^3，48スレッド N= 2,097,152

	Fortran	C (clang)	C (trad)
src0（初期設定）	1.671	1.564	2.354
src1（First Touch）	1.480	1.122	1.720
src2（+ELL）	0.747	0.809	1.127
src3（+omp-parallel削減）	0.707	0.834	0.854
src3b（+ clang loop unroll_count）	-	0.764	-
src4 (+omp-do無し，reduction無し)	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のみ

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

PCG法：計算時間：N=160^3，48スレッド N= 4,096,000

	Fortran	C (clang)	C (trad)
src0（初期設定）	3.610	3.484	4.067
src1（First Touch）	2.993	2.228	3.425
src2（+ELL）	1.534	1.690	2.340
src3（+omp-parallel削減）	1.556	1.693	1.742
src3b（+ clang loop unroll_count）	-	1.586	-
src4 (+omp-do無し，reduction無し)	1.435	-	-

PCG法：計算時間：N=200^3，48スレッド N= 8,000,000

	Fortran	C (clang)	C (trad)
src0（初期設定）	7.666	8.321	9.397
src1（First Touch）	6.952	5.102	8.008
src2（+ELL）	3.421	3.910	5.381
src3 （+omp-parallel削減）	3.440	3.920	3.824
src3b （+ clang loop unroll_count）	-	3.624	-
src4 (+omp-do無し，reduction無し)	3.276	-	-

PCG法：計算時間：N=256^3，48スレッド N= 16,777,216

	Fortran	C (clang)	C (trad)
src0（初期設定）	34.308	24.772	25.547
src1（First Touch）	32.202	22.172	23.814
src2（+ELL）	8.916	10.761	14.566
src3（+omp-parallel削減）	8.915	10.764	10.415
src3b（+ clang loop unroll_count）	-	10.003	-
src4 (+omp-do無し，reduction無し)	8.620	-	-

まとめ

- 有限体積法によるポアソン方程式ソルバー，PCG法による連立一次方程式求解⇒OpenMP並列化
- Wisteria/BDEC-01（Odyssey）（A64FX）による実習
  - 複数CMGを使う
  - Cコンパイラの挙動⇒2021年11月時点より改善
- 様々な最適化
  - First Touch Data Placement
  - ELL，OpenMPオーバーヘッド削減
- 一日速習に丁度よい分量，であると考えている
  - アンケートで忌憚のないご意見をお願いいたします
- データ依存性を含む場合（ICCG）は，3日間，2022年秋以降に開催予定

性能評価：詳細プロファイラ

- 測定部をプログラム内で指定（複数設定可）
- 17回実行
- 計算性能，消費電力等詳細なデータを得られる
- Excel（マクロ）
  - https://www.dropbox.com/s/kat9ny5aoxp7cqm/cpu_pa_report.xlsm?dl=0

詳細プロファイラ利用（1/4）

測定部分をプログラム内で指定，特殊なコンパイルオプション不要：solver_PCG.c/f

```c
#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 {
           RHO1 = RHO;
        }
}

        *IER = 1;
N900:
        Etime = omp_get_wtime();
        fapp_stop ("CG",1,0);

        return 0;
}
```

```fortran
       call fapp_start ("CG",1,0)
       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()
       call fapp_stop ("CG",1,0)

       return
       end
```

詳細プロファイラ利用（2/4）
Wisteria/BDEC-01上での実行

```
>$ cd /work/gt00/t00XYZ/ompw
>$ cd run

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

>$ pjsub fapp.sh

(after finishing)

>$ pjsub data.sh

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

詳細プロファイラ利用（3/4）

Wisteria/BDEC-01上での実行スクリプト

各ディレクトリが既に生成されている場合は中身を空にしておくこと（上書きはしない）

fapp.sh

```
#!/bin/sh
#PJM -N "fapp"
#PJM -L rscgrp=tutorial-o
#PJM -L node=1
#PJM -omp thread=48
#PJM -L elapse=00:15:00
#PJM -g gt00
#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=lecture-o
#PJM -L node=1
#PJM --mpi proc=1
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o data.lst

module load fj
module load fjmpi

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
```

詳細プロファイラ利用（4/4）
PC上での実行

- pa*.csvをPC上へコピー

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

- 上記pa*.csvとExcelマクロファイルを同じディレクトリに置く
- Excelマクロファイルをダブルクリック
  - 指示に従う
  - AllではなくCGを選択

PCG法：計算時間：N=160^3，48スレッド

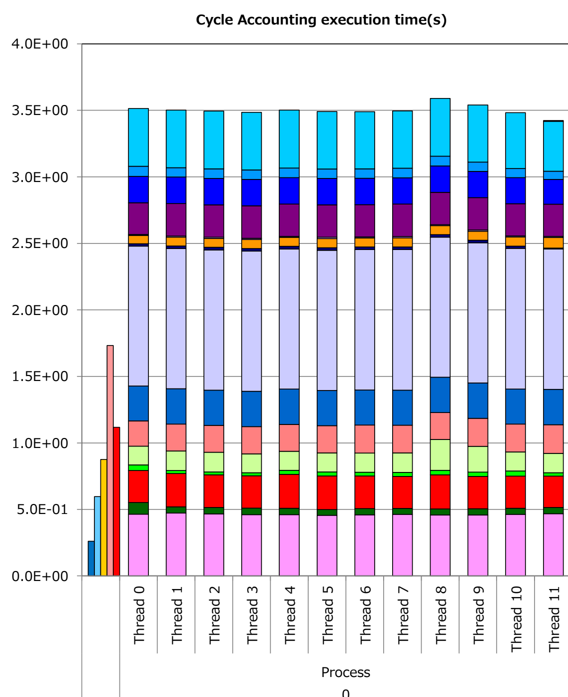
N= 4,096,000，ノード当たり換算，Fortran

最適化⇒メモリビジー⇒消費電力増加，Joule値は低下

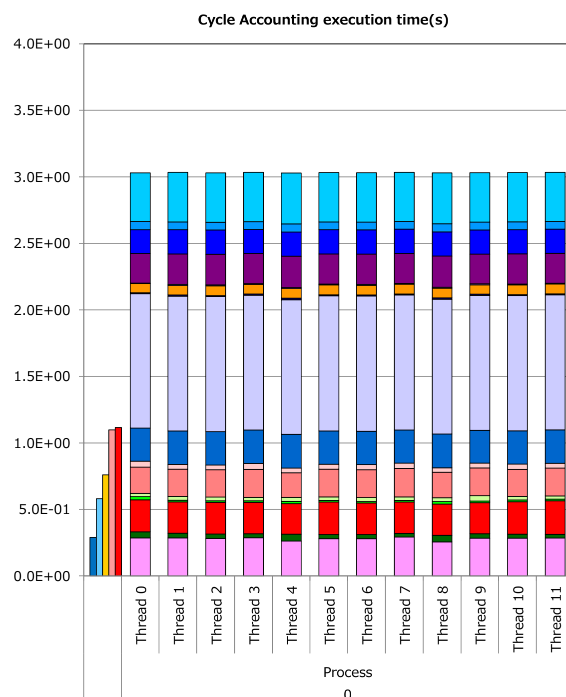
	計算時間 (sec.)	対ピーク性能比 (%)	SIMD化率 (%)	Memory Throughput (%)	Instruction		Power (W)	
					Effective	Load/ Store	Core L2 Memory	Node
solf0	3.69	1.59	20.0	30.3	3.39×10^{11}	8.27×10^{10}	81.6 10.9 20.2	112.
solf1 (First Touch)	2.98	1.97	28.8	37.5	2.35×10^{11}	5.33×10^{10}	92.1 10.8 33.3	136.
solf2 (+ ELL)	1.58	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法：性能評価：N=160^3，48スレッド Fortran，各CMG

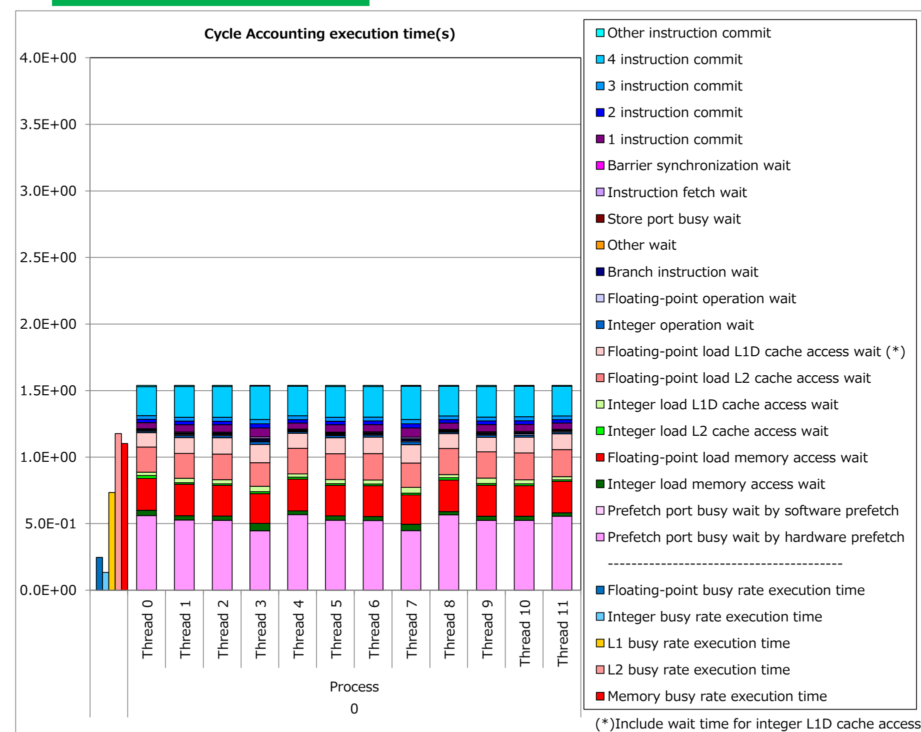
src0:Initial



src1:First Touch



src2: +ELL



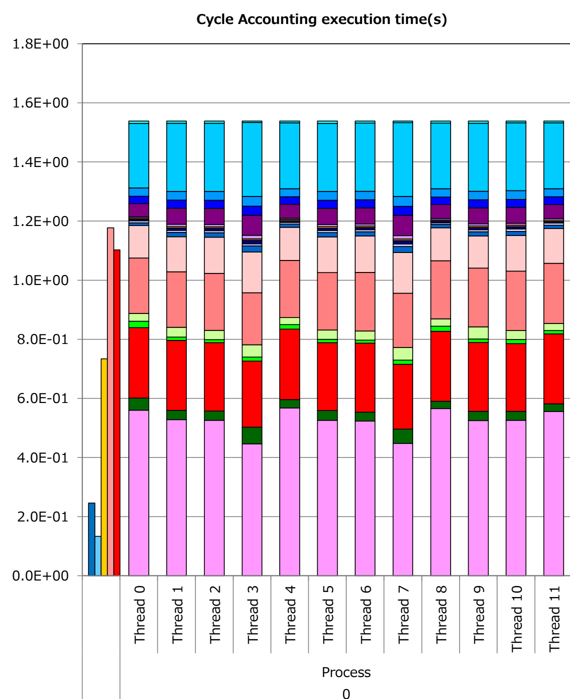
- **Prefetch Port Busy Wait (H/W)**
- **Integer Load L1D Cache Access Wait**
- **Floating-Point Operation Wait**
- **Instruction Commit**

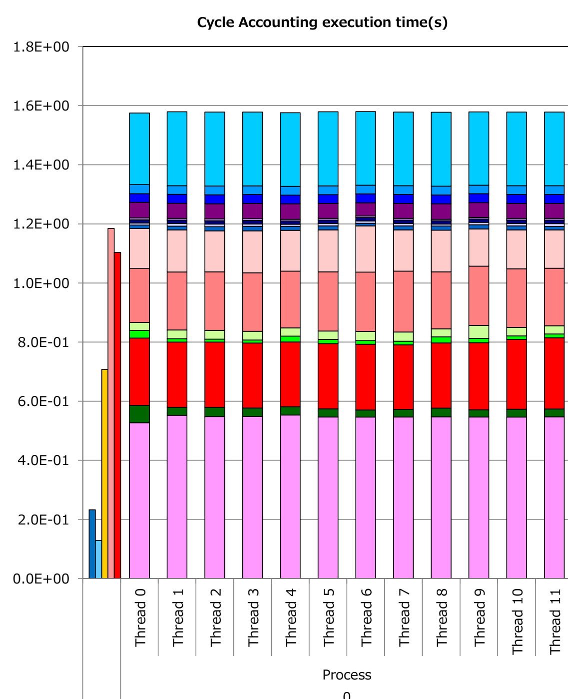
PCG法：性能評価：N=160^3，48スレッド Fortran，各CMG

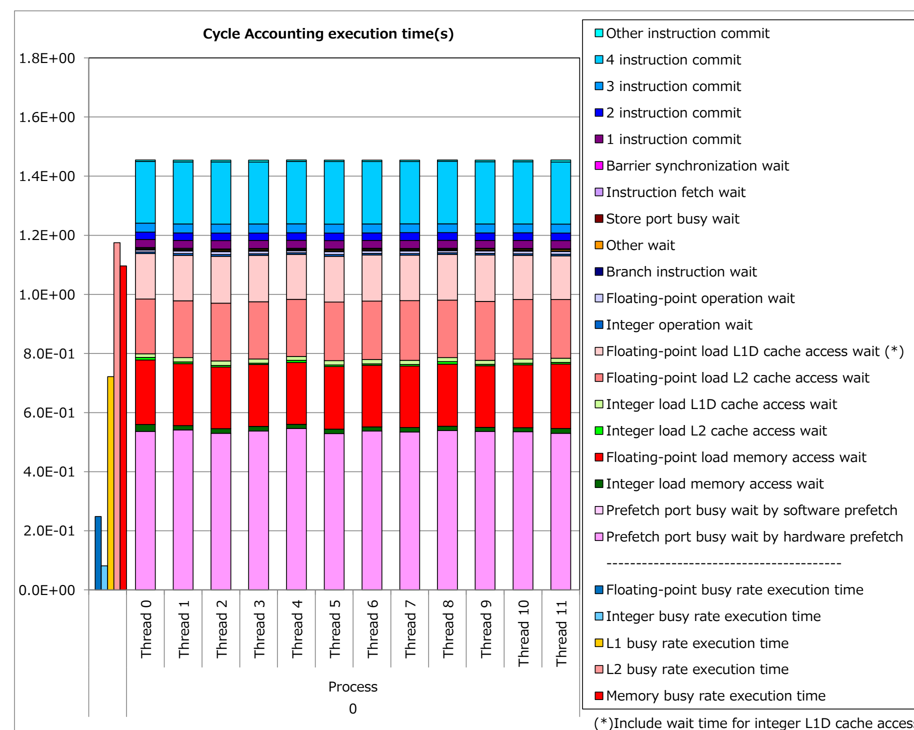
src2:ELL

src3:omp-parallel削減

src4: omp-do無し，reduction無し







- **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**

PCG法：計算時間：N=160^3，48スレッド

N= 4,096,000，ノード当たり換算，C（clang）

最適化⇒メモリビジー⇒消費電力増加，Joule値は低下

	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.

PCG法：計算時間：N=160^3，48スレッド

N= 4,096,000，ノード当たり換算，C（trad）

最適化⇒メモリビジー⇒消費電力増加，Joule値は低下

	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.