科学技術計算のための
マルチコアプログラミング入門
演習編

中島研吾

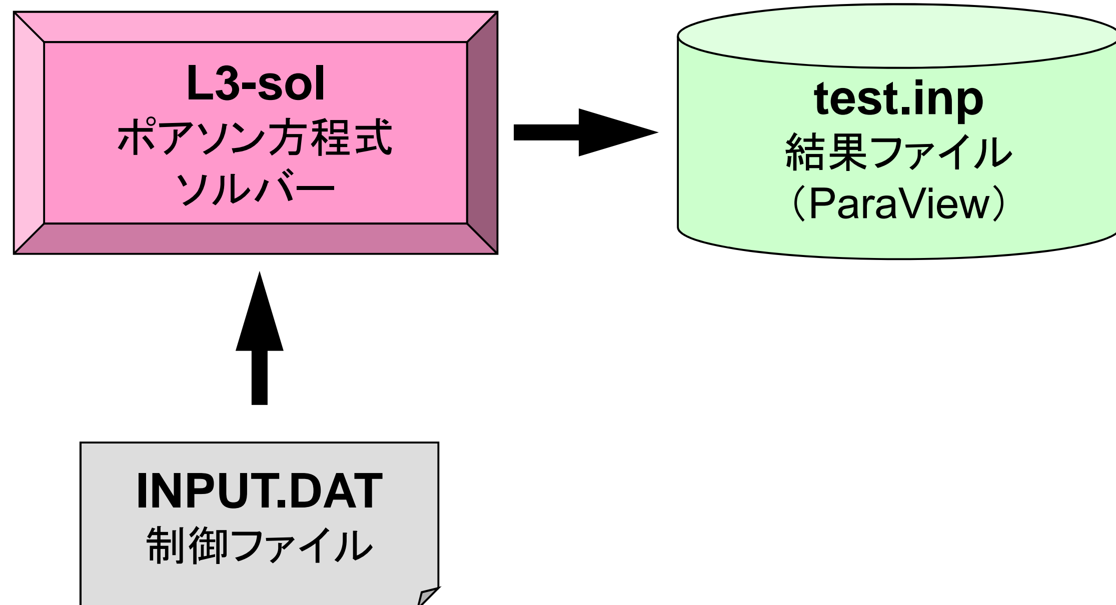
東京大学情報基盤センター

- マルチコア版コードの実行
- 更なる最適化
- STREAM
- コンパイルオプション
  - FORTRANのみ
  - コンパイルリストの分析

コンパイル・実行

```
>$ cd <$O-L3>/src
>$ make
>$ ls ../run/L3-sol

  L3-sol

>$ cd ../run

>$ pjsub go1.sh
```

プログラムの実行
プログラム，必要なファイル等



制御データ（INPUT.DAT）

```
100 100 100                             NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00              DX/DY/DZ
1.0e-08                                 EPSICCG
16                                      PEsmpTOT
-10                                     NCOLORtot
```

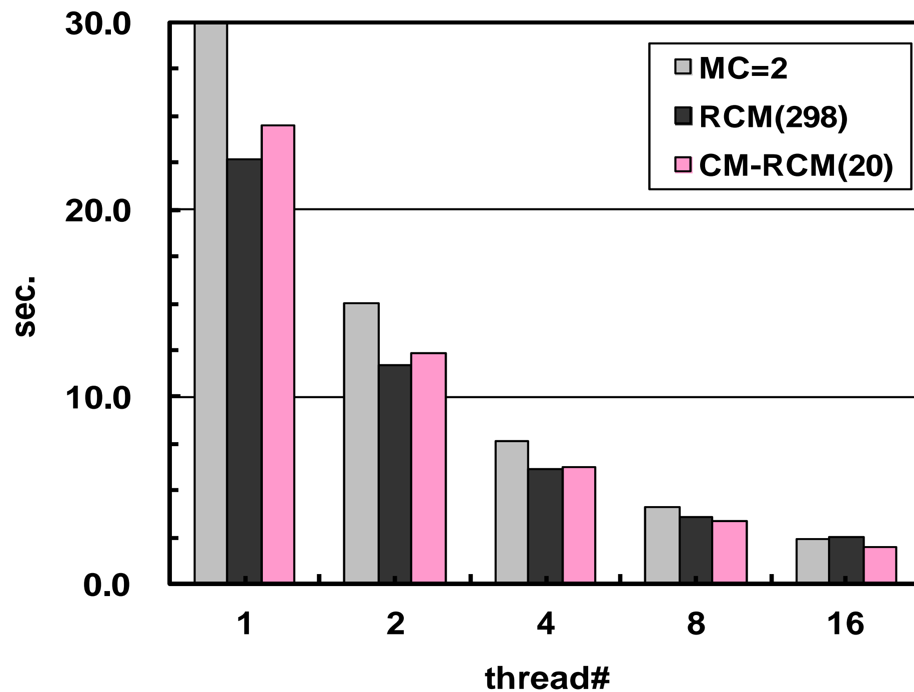
変数名	型	内　容
NX，NY，NZ	整数	各方向の要素数
DX，DY，DZ	倍精度実数	各要素の3辺の長さ（ΔX，ΔY，ΔZ）
EPSICCG	倍精度実数	収束判定値
PEsmpTOT	整数	データ分割数
NCOLORtot	整数	Ordering手法と色数 ≧2：MC法（multicolor），色数 =0：CM法（Cuthill-Mckee） =-1：RCM法（Reverse Cuthill-Mckee） ≦-2：CM-RCM法

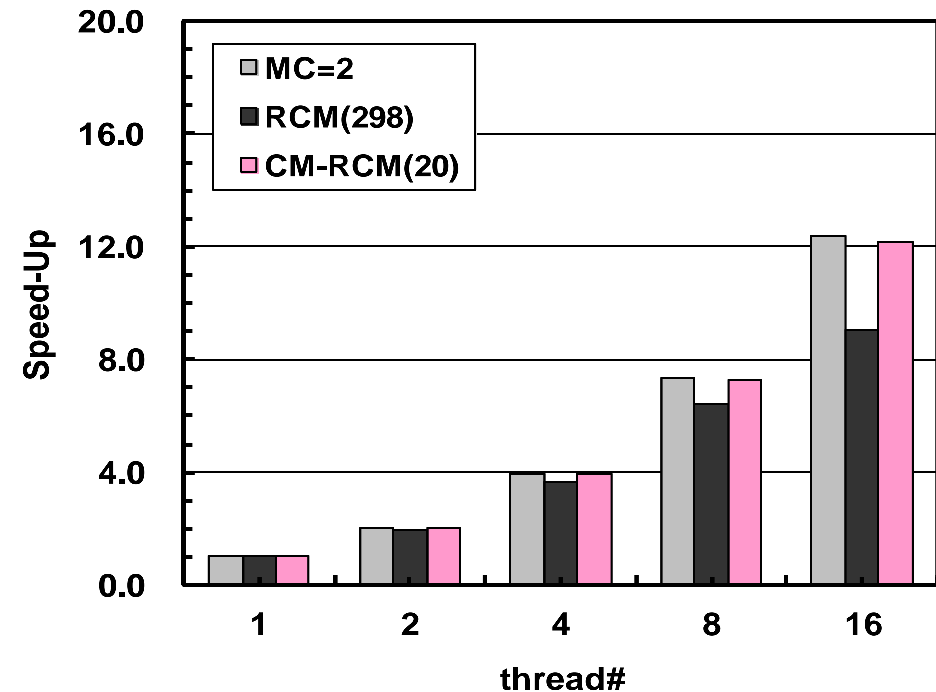
go1.sh

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:10:00"
#PJM -L "rscgrp=lecture"
#PJM -g "gt00"
#PJM -j
#PJM -o "test.lst"                標準出力ファイル名

export OMP_NUM_THREADS=16         スレッド数，通常 =PEsmpTOT
./L3-sol
```

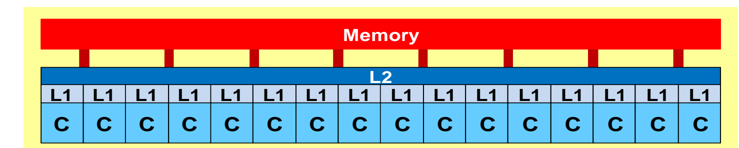
計算結果（FX10@東大）：10^6要素

反復回数：MC（2色）：333回，RCM（298レベル）：224回 CM-RCM（Nc=20）：249回





16 threads
MC(2): 2.42 sec.
CM-RCM(20): 2.01 sec.



実習(1)

- 色々なケースでやってみよう
  - 問題サイズ
  - スレッド数
  - 色数，色分け法（MC, RCM, CM-RCM）

- マルチコア版コードの実行
- 更なる最適化
- STREAM
- コンパイルオプション
  - FORTRANのみ
  - コンパイルリストの分析

前進代入：現状の並列化（Fortran）

```
      do ic= 1, NCOLORtot
!$omp parallel do private(ip,ip1,i,WVAL,k)
        do ip= 1, PEsmpTOT
          ip1= (ic-1)*PEsmpTOT + ip
        do i= SMPindex(ip1-1)+1, SMPindex(ip1)
          WVAL= W(i,Z)
          do k= indexL(i-1)+1, indexL(i)
            WVAL= WVAL -  AL(k) * W(itemL(k),Z)
          enddo
          W(i,Z)= WVAL * W(i,DD)
        enddo
        enddo
!$omp end parallel do
      enddo
```

- 「!omp parallel」でスレッド（～16）の生成，消滅が発生
  - 色ごとにこの部分を通る
  - 多少のオーバーヘッドがある
- 色数が増えるとオーバーヘッドが増す

前進代入：現状の並列化(C)

```
for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for private (ip, ip1, i, WVAL, j)
        for(ip=0; ip<PEsmpTOT; ip++) {
                ip1 = ic * PEsmpTOT + ip;
                for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++){
                        WVAL = W[Z][i];
                        for(j=indexL[i]; j<indexL[i+1]; j++){
                                WVAL -= AL[j] * W[Z][itemL[j]-1];
                        }
                        W[Z][i] = WVAL * W[DD][i];
                }
        }
}
```

- 「#pragma omp parallel」でスレッド(～16)の生成，消滅が発生
  - 色ごとにこの部分を通る
  - 多少のオーバーヘッドがある
- 色数が増えるとオーバーヘッドが増す

前進代入：Overhead削減（Fortran）

```
!$omp parallel private(ip,ip1,i,WVAL,k)
      do ic= 1, NCOLORtot
!$omp do
        do ip= 1, PEsmpTOT
          ip1= (ic-1)*PEsmpTOT + ip
        do i= SMPindex(ip1-1)+1, SMPindex(ip1)
          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
      endd
!$omp end parallel
```

- このようにすることによって，スレッド生成を前進代入に入る前の一回で済ませることができる
- 「!omp do」のループが並列化

前進代入：Overhead削減（C）

```
#pragma omp parallel private (ip, ip1, i, WVAL, j)
for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp for
        for(ip=0; ip<PEsmpTOT; ip++) {
                ip1 = ic * PEsmpTOT + ip;
                for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++){
                        WVAL = W[Z][i];
                        for(j=indexL[i]; j<indexL[i+1]; j++){
                                WVAL -= AL[j] * W[Z][itemL[j]-1];
                        }
                        W[Z][i] = WVAL * W[DD][i];
                }
        }
}
```

- このようにすることによって，スレッド生成を前進代入に入る前の一回で済ませることができる
- 「#pragma omp for」のループが並列化

プログラム類

```
% cd <$O-L3>
% ls
    run   short src  src0

    ※short (Fortranのみ)

% cd src0

% make
% cd ../run
% ls L3-sol0
    L3-sol0

% <modify “INPUT.DAT”>
% <modify “go0.sh”>

% pjsub go0.sh
```

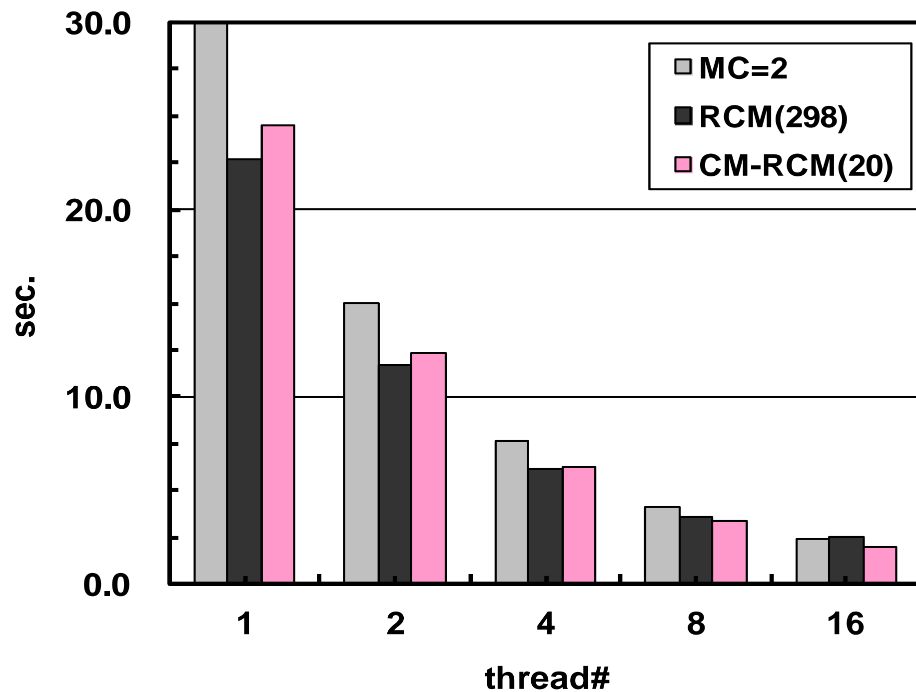
計算結果：色数が多いと少し速い $N=128^3$

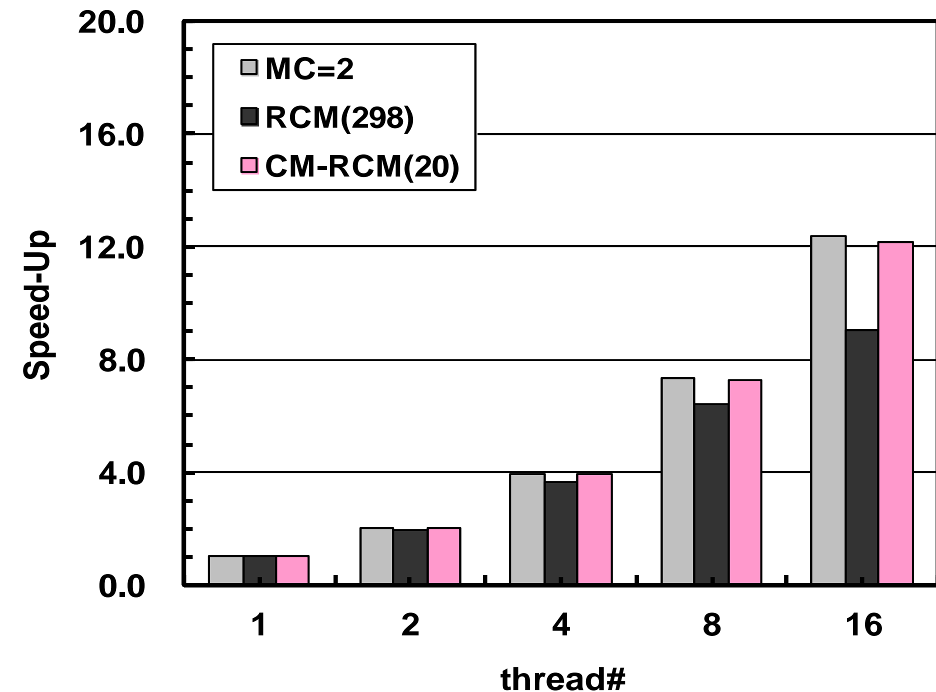
	L3-sol	L3-sol-0
NCOLORtot= -20 CM-RCM (20) 318 Iterations	5.69 sec.	5.67 sec.
NCOLORtot= -1 RCM (382 levels) 287 Iterations	6.54 sec.	6.38 sec.

- マルチコア版コードの実行
- 更なる最適化
- **STREAM**
- コンパイルオプション
  - FORTRANのみ
  - コンパイルリストの分析

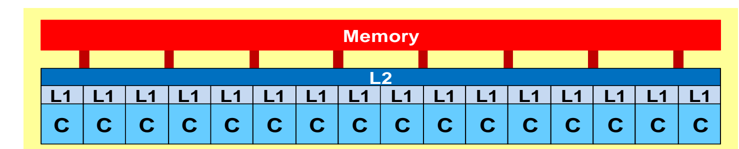
計算結果（FX10@東大）：10^6要素

反復回数：MC（2色）：333回，RCM（298レベル）：224回 CM-RCM（Nc=20）：249回





16 threads
MC(2): 2.42 sec.
CM-RCM(20): 2.01 sec.



何故16倍にならないか？

- メモリ競合
- 16スレッドがメモリにアクセスすると，1スレッドの場合と比較して，スレッド当り（コア当り）メモリ性能は低下
- 疎行列はmemory-boundなためその傾向がより顕著
  - 疎行列計算の高速化：研究途上の課題
- 問題規模が比較的小さい

疎行列：非零成分のみ記憶
⇒メモリへの負担大
（memory-bound）：間接参照
（差分，FEM，FVM）

```
{Y}= [A]{X}

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

行列ベクトル積：密行列⇒とても簡単 メモリへの負担も小さい

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,N-1} & a_{1,N} \\ a_{21} & a_{22} & & a_{2,N-1} & a_{2,N} \\ \cdots & & \cdots & & \cdots \\ a_{N-1,1} & a_{N-1,2} & & a_{N-1,N-1} & a_{N-1,N} \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N-1} & a_{N,N} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{Bmatrix}$$

```
{Y}= [A]{X}

do j= 1, N
  Y(j)= 0.d0
  do i= 1, N
    Y(j)= Y(j) + A(i,j)*X(i)
  enddo
enddo
```

GeoFEM Benchmark

ICCG法の性能（固体力学向け）

	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

疎行列ソルバー：Memory-Bound

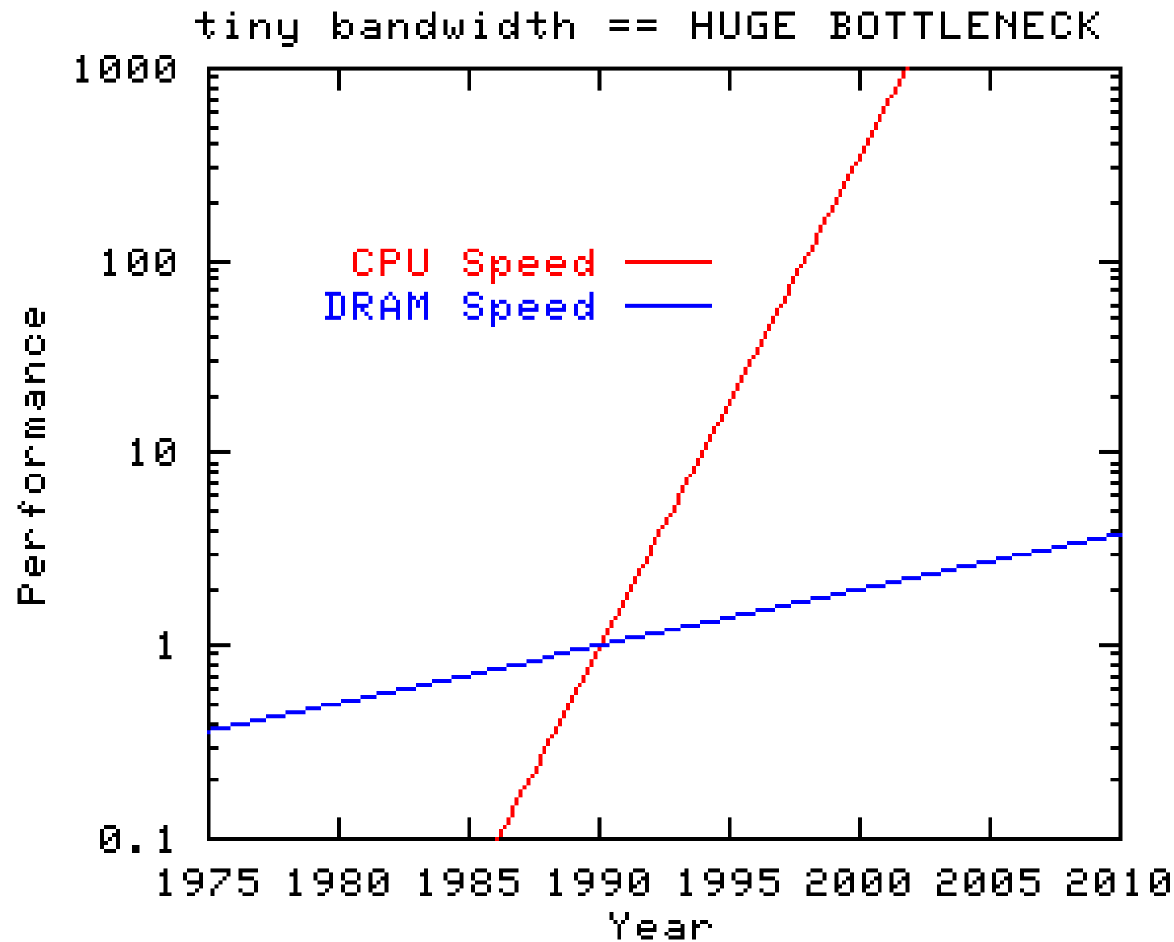
STREAM ベンチマーク

http://www.streambench.org/

- メモリバンド幅を測定するベンチマーク
  - 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
```

マイクロプロセッサの動向 CPU性能，メモリバンド幅のギャップ



http://www.streambench.org/

実行：OpenMPバージョン

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

run.sh

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

export PATH=…
export LD_LIBRARY_PATH=…
export PARALLEL=16
export OMP_NUM_THREADS=16                スレッド数（1-16）

./stream.out > 16-01.lst 2>&1           出力ファイル名
```

Triadの結果 <$O-stream>/*.lst

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

実習(2)

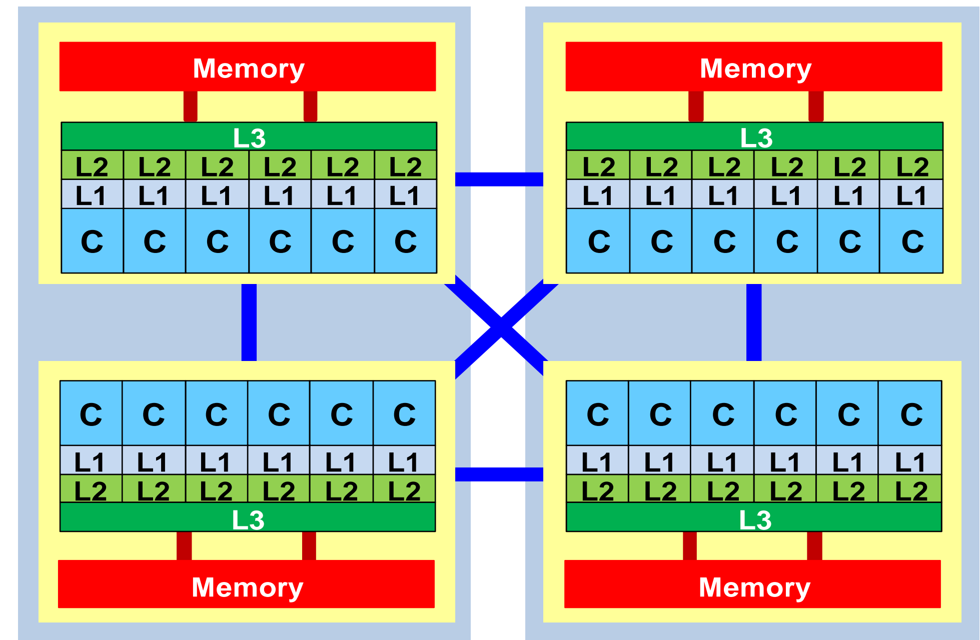
- 実際にやってみよ
- スレッド数を変える
- 1CPU版, MPI版もある
  - FORTRAN, C
  - STREAMのサイト

- マルチコア版コードの実行
- 更なる最適化
- STREAM
- **コンパイルオプション**
  - **FORTRANのみ**
  - **コンパイルリストの分析**

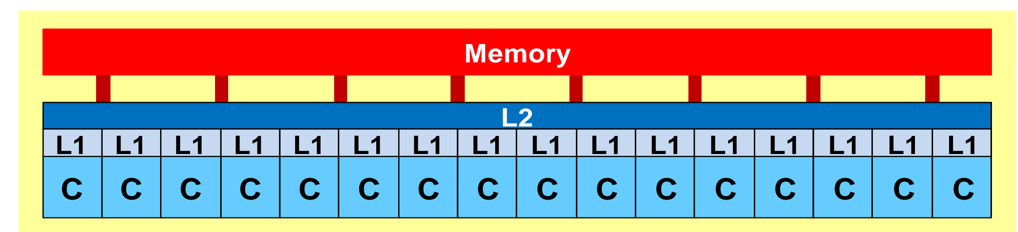
ccNUMA Architecture

- Multicore-Multisocket
- 各ソケットがローカルにメモリを持っている
  - cc-NUMA：cache-coherent-Non-Uniform Memory Access
  - ローカルのメモリをアクセスして計算するようなプログラミング，データ配置，実行時制御（numactl）が必要
  - T2K，Cray XE6 etc.
- 特殊な最適化
  - First-Touch Data Placement
  - Sequential Reordering
- 京，FX10 → Flat，UMA

Cray XE6 (Hopper)



Fujitsu FX10 (Oakleaf-FX)

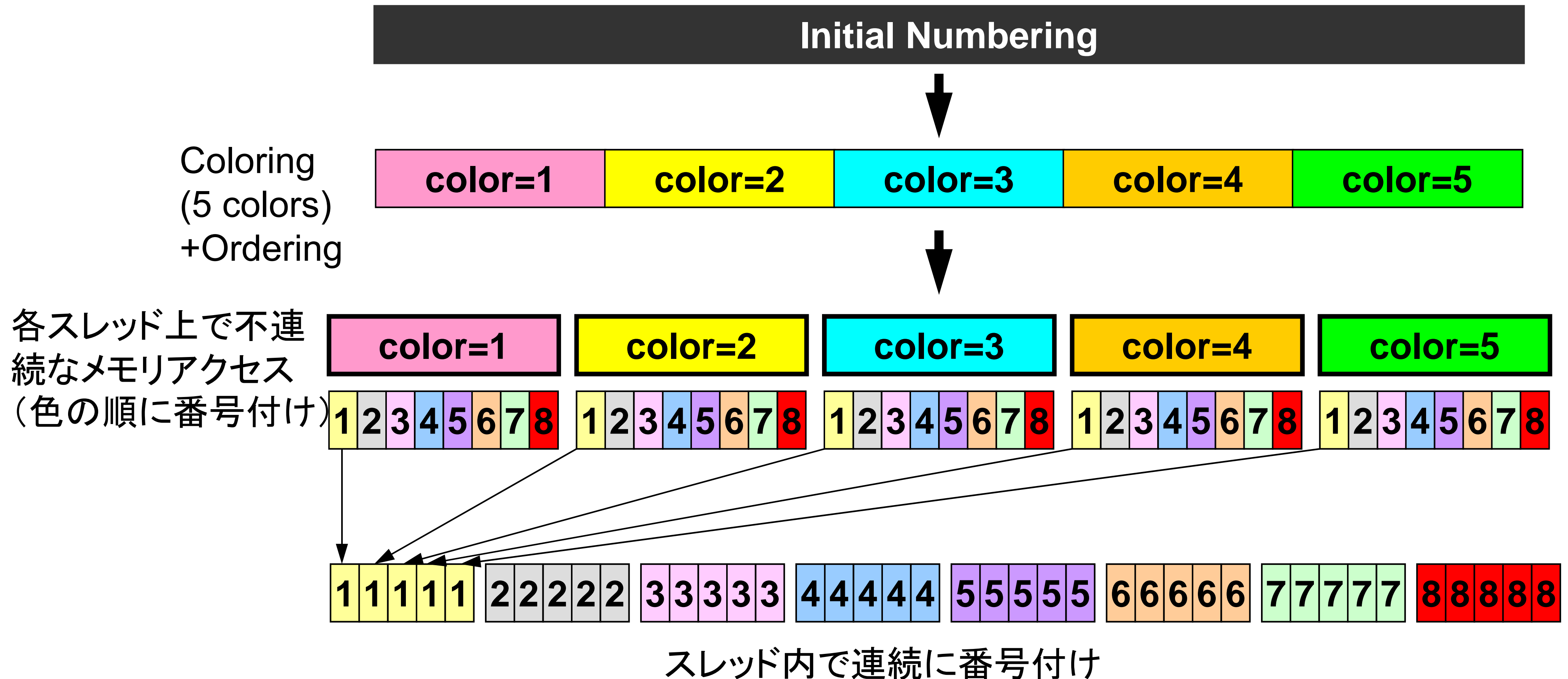


First Touch Data Placement

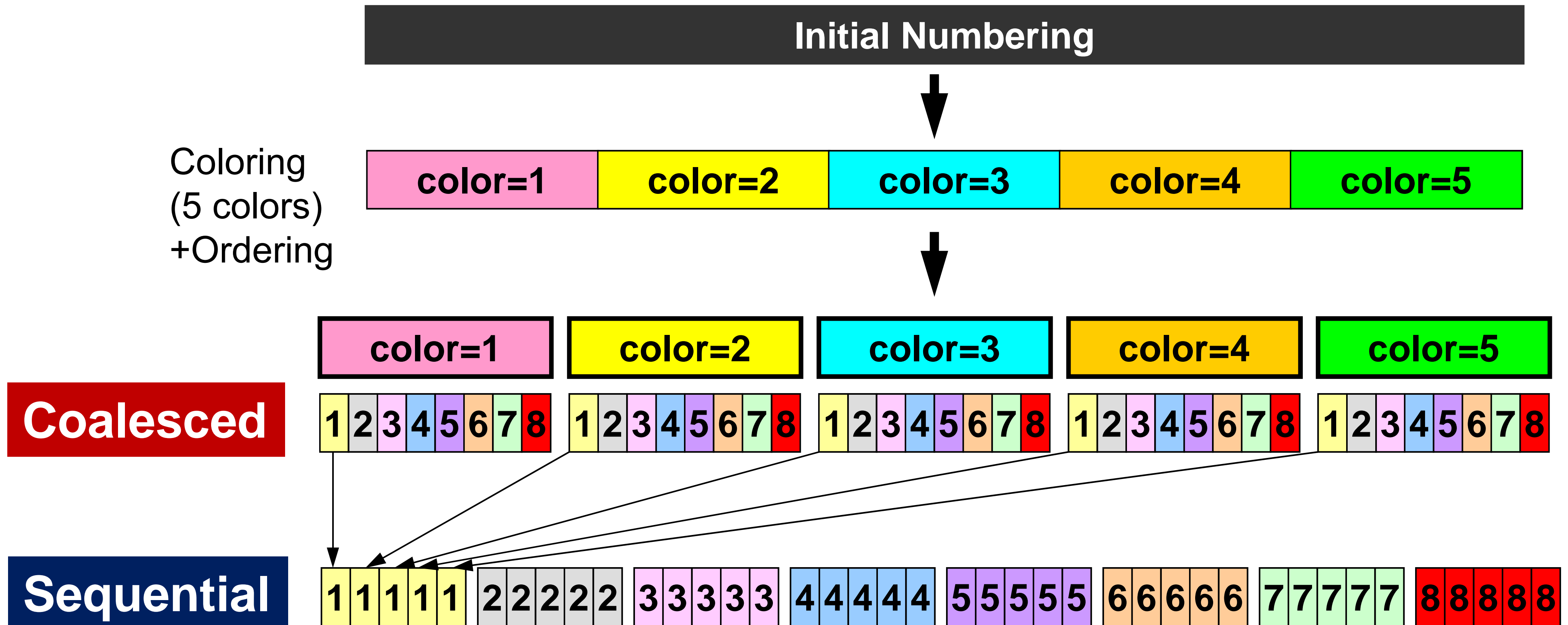
配列のメモリ・ページ：
最初にtouchしたコアのローカルメモリ上に確保
計算と同じ順番で初期化

```
  do lev= 1, LEVELtot
    do ic= 1, COLORtot(lev)
!$omp parallel do private(ip,i,j,isL,ieL,isU,ieU)
      do ip= 1, PEsmpTOT
      do i = STACKmc(ip,ic-1,lev)+1, STACKmc(ip,ic,lev)
        RHS(i)= 0.d0; X(i)= 0.d0; D(i)= 0.d0

        isL= indexL(i-1)+1
        ieL= indexL(i)
        do j= isL, ieL
          itemL(j)= 0; AL(j)= 0.d0
        enddo

        isU= indexU(i-1)+1
        ieU= indexU(i)
        do j= isU, ieU
          itemU(j)= 0; AU(j)= 0.d0
        enddo
      enddo
      enddo
!$omp omp end parallel do
    enddo
  enddo
```

各スレッド上でメモリアクセスが連続となるように更なる並び替え，「sequential」 5 colors, 8 threads



各スレッド上でメモリアクセスが連続となるように更なる並び替え，「sequential」 5 colors, 8 threads



Hopper & Oakleaf-FX

	Cray XE6 Hopper	FX10 Oakleaf-FX
Peak Performance/core (GFLOPS)	8.40	14.78
Core #/Node	24	16
Peak Performance/node (GFLOPS)	201.6	236.5
Peak Memory Bandwidth/node (GB/sec)	85.3 8xDDR3 1333MHz	85.3 8xDDR3 1333MHz
Triad Performance/node (GB/sec)	52.3	64.7
Triad Performance/core (GB/sec)	2.18	4.04
B/F Rate	0.260	0.274
GeoFEM Bench/ICCG (MFLOPS/core)	469.8	1011.3
GeoFEM Bench/ICCG (GFLOPS/node)	11.28	16.18

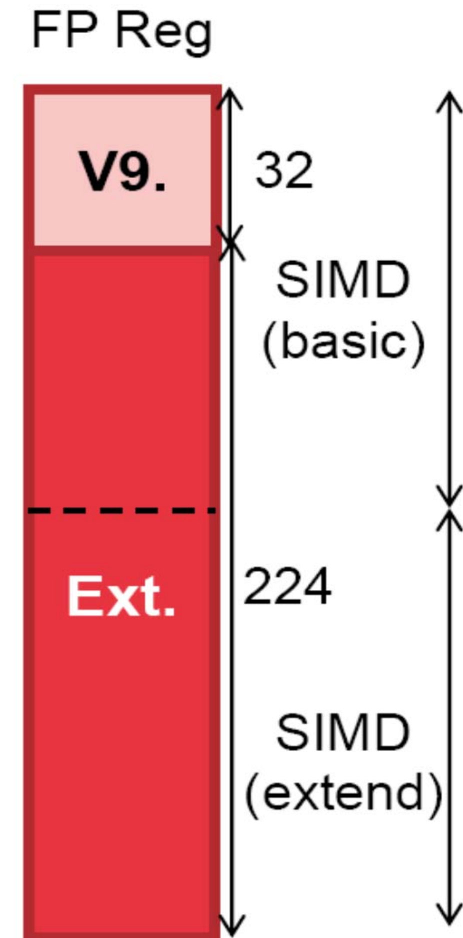
Optimizations for ccNUMA are not effective on Oakleaf-FX

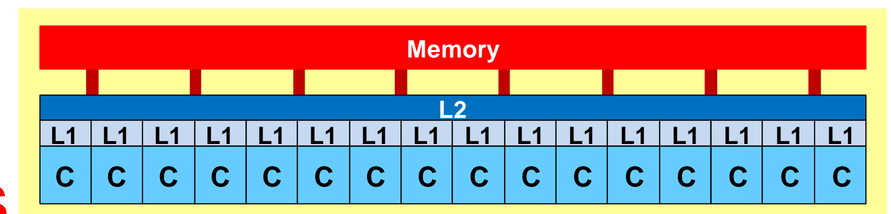
Poisson's Equations, ICCG/CM-RCM(2), 128^3 unknows
Time for ICCG solver until convergence (318 iter's).

	Cray XE6 (Hopper)	Oakleaf-FX (not optimum compiler options)
Original	18.1 sec	5.62 sec
+ First Touch	7.81 sec	5.94 sec
+ Sequential Reordering	5.04 sec	5.84 sec

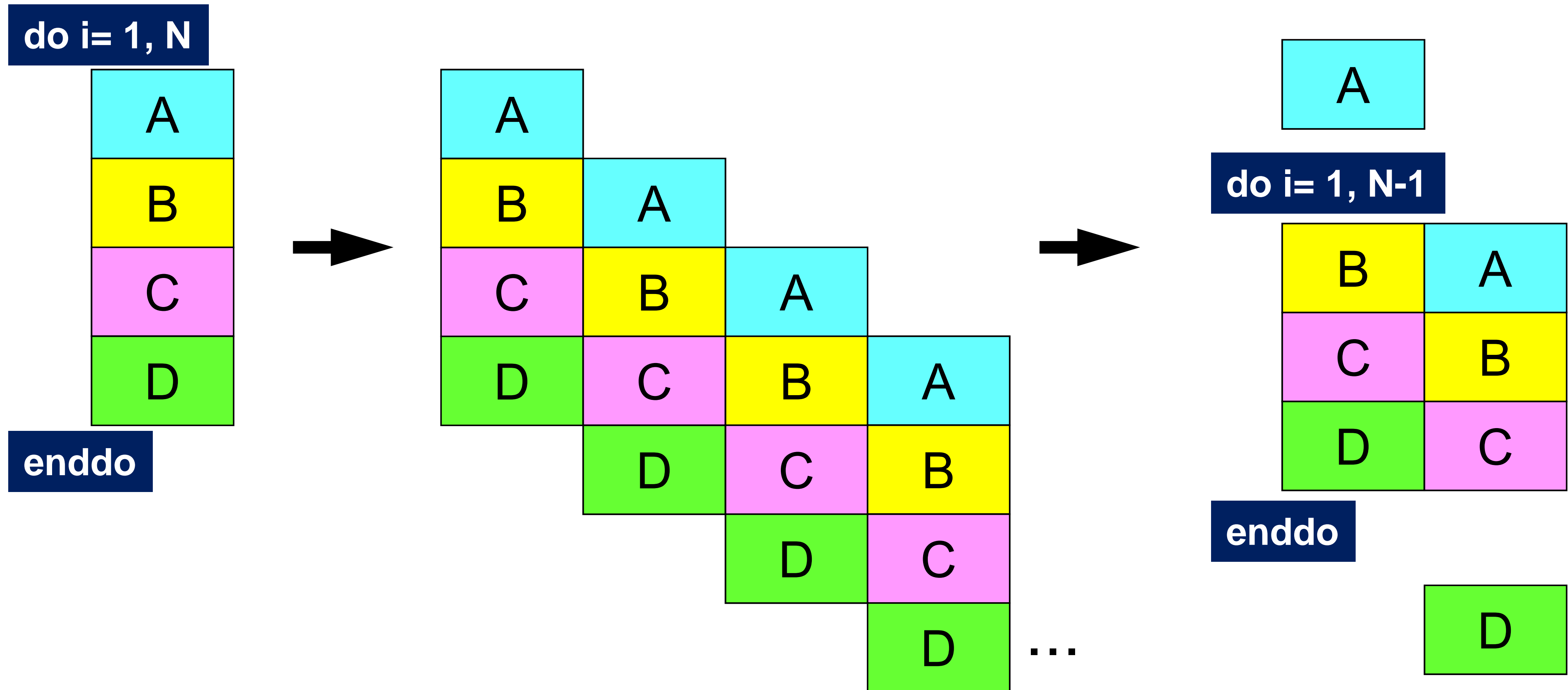
SPARC64™ IXfx

- HPC-ACE (High Performance Computing – Arithmetic Computational Extensions)
  - Enhanced instruction set for the SPARC-V9 instruction set arch.
    - High-Performance & Power-Aware
  - Extended number of registers
    - FP Registers: 32→256
    - Software Pipelining is useful
  - S/W controllable “sector” cache
- UMA, not NUMA
- H/W barrier for high-speed synchronization of on-chip cores





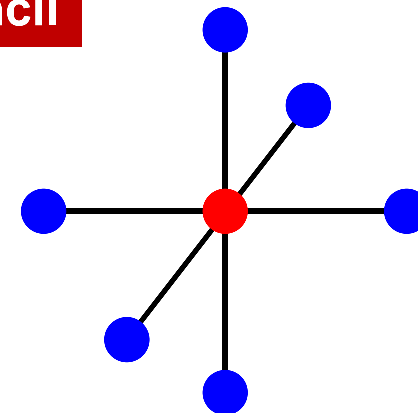
Pipelining + SIMD

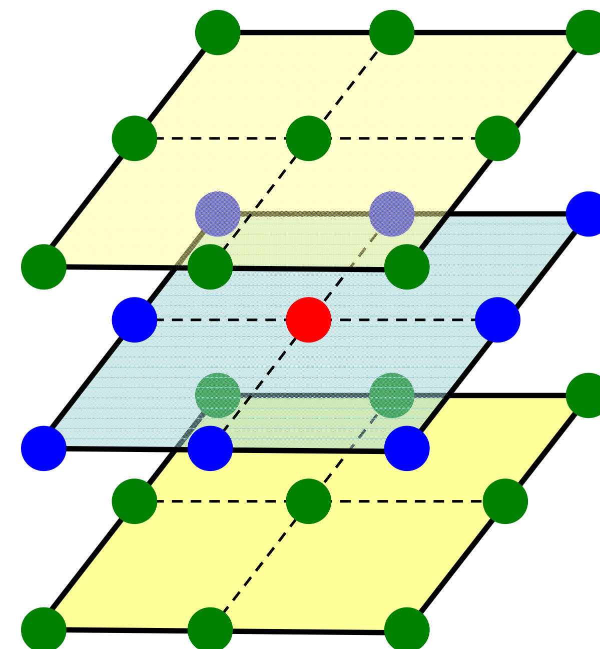


もしこのような実行が可能であれば･･･

現行コードの問題

- 最内側ループの回転数（ループ長）が短い
  - いわゆる7点ステンシル，最大でも6
  - 大小関係を考慮する上下三角成分では色数が少ない場合は6に近いが，色数が増えると3程度
  - Software Pipeliningが効きにくい
- 有限要素法（GeoFEM Benchmark）だと性能が出やすい
  - 27点ステンシル





Hopper & Oakleaf-FX: 27-pt. stencil

	Hopper	Oakleaf FX
Peak Performance/core (GFLOPS)	8.40	14.78
Core #/Node	24	16
Peak Performance/node (GFLOPS)	**201.6**	**236.5**
Peak Memory Bandwidth/node (GB/sec)	85.3 8xDDR3 1333MHz	85.3 8xDDR3 1333MHz
Triad Performance/node (GB/sec)	**52.3**	**64.7**
Triad Performance/core (GB/sec)	2.18	4.04
B/F Rate	0.260	0.274
GeoFEM Bench/ICCG (GFLOPS/node)	**11.28**	**16.18**
GeoFEM Bench/ICCG (MFLOPS/core)	469.8	1011.3

最内ループ回転数が少ない場合

- コンパイラ：専用オプション（Fortranのみ）
  - -Kshortloop= N (N:2-10), -Knoshortloop（default）
- 最内側ループの強制Unrolling（N/2段）
- 以下は抑止
  - ソフトウェアパイプライニング
  - ループブロッキング
  - ループストライピング
- 回転数が10より大きいときに指定すると性能が低下する場合あり
- 結果（128^3, CM-RCM(20)） on FX10@東大
  - Default: 5.62 sec.
  - Kshortloop: 5.39 sec. (N=2), 5.39 sec. (N=3), 5.61 sec. (N=4)
    - それでもCray XE6より遅い

コンパイル・実行

デフォルト

```
>$ cd <$O-L3>/src
>$ make
>$ ls ../run/L3-sol
   L3-sol
>$ cd ../run
>$ pjsub go1.sh
```

```
F90          = frtpx
F90OPTFLAGS= -Kfast,openmp -Qt
F90FLAGS =$(F90OPTFLAGS)
```

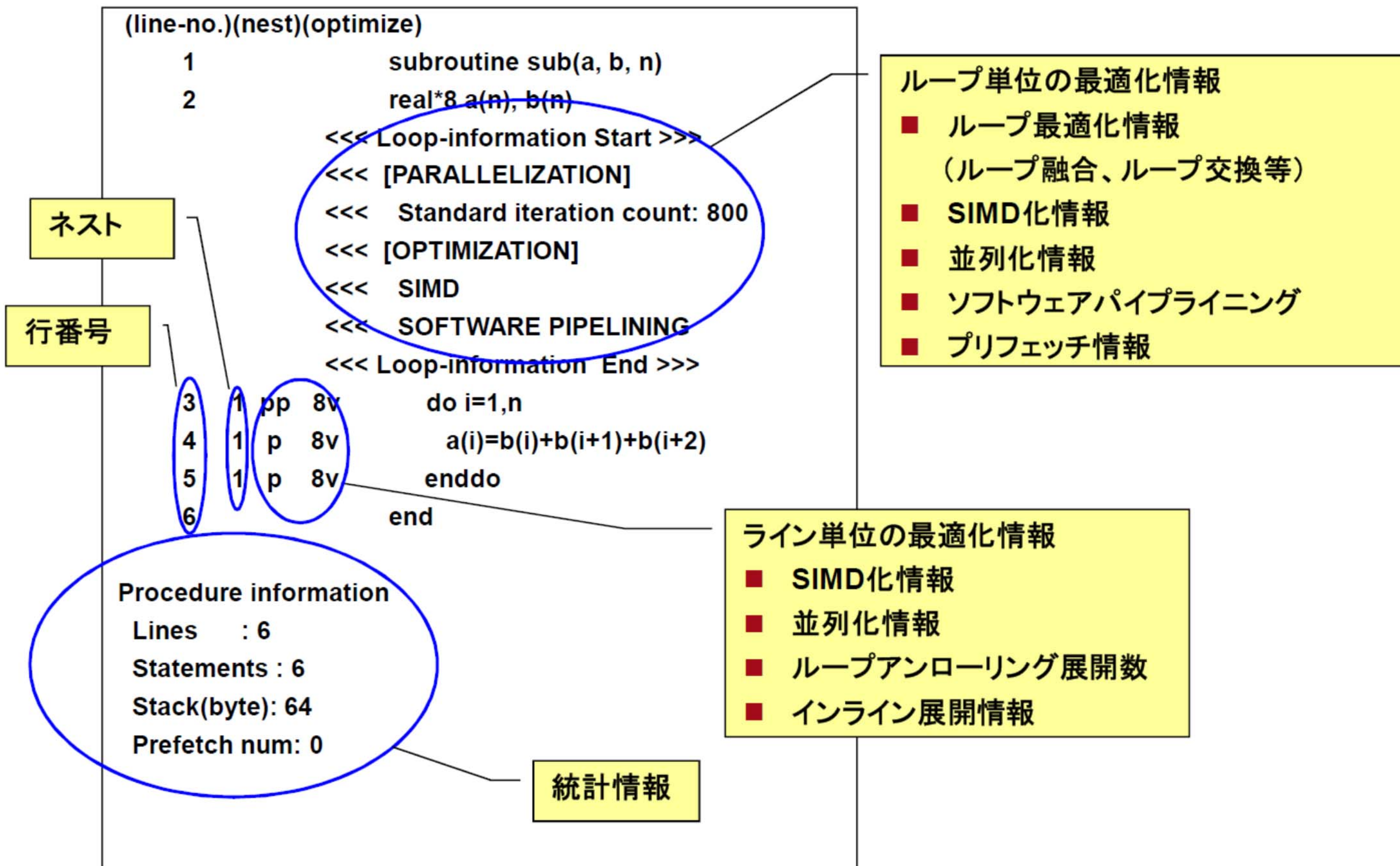
Kshortloop

```
>$ cd <$O-L3>/short
>$ make
>$ ls ../run/L3-sols
   L3-sols
>$ cd ../run
>$ pjsub gos.sh
```

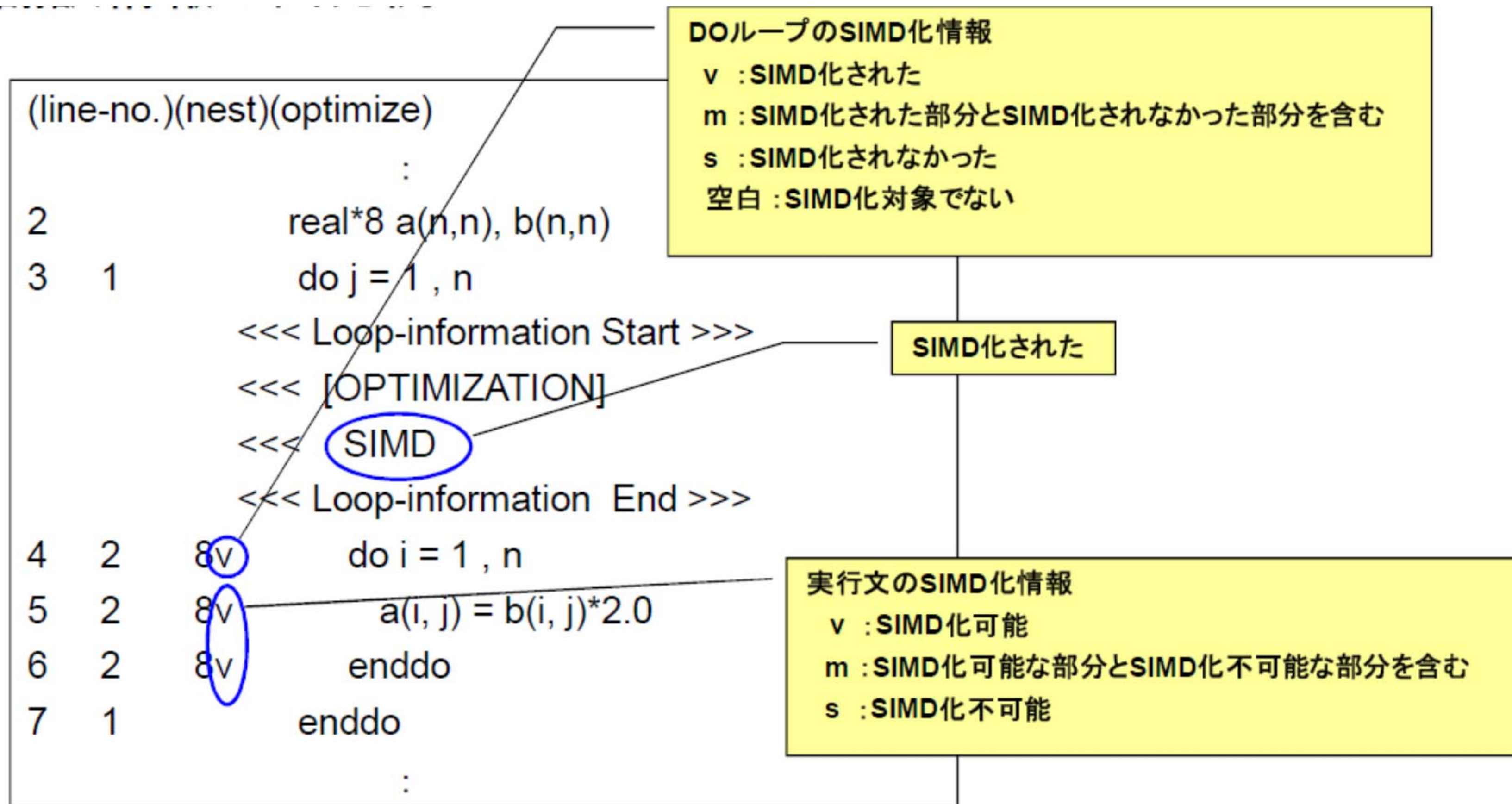
```
F90          = frtpx
F90OPTFLAGS= -Kfast,openmp -Kshortloop=3 -Qt
F90FLAGS =$(F90OPTFLAGS)
```

- -Qt
  - コンパイルリスト出力
  - ＊.lst
- Cでは「-Qt」は使えません，「-Nsrc」を使ってください。
  - 画面に出てしまいますが

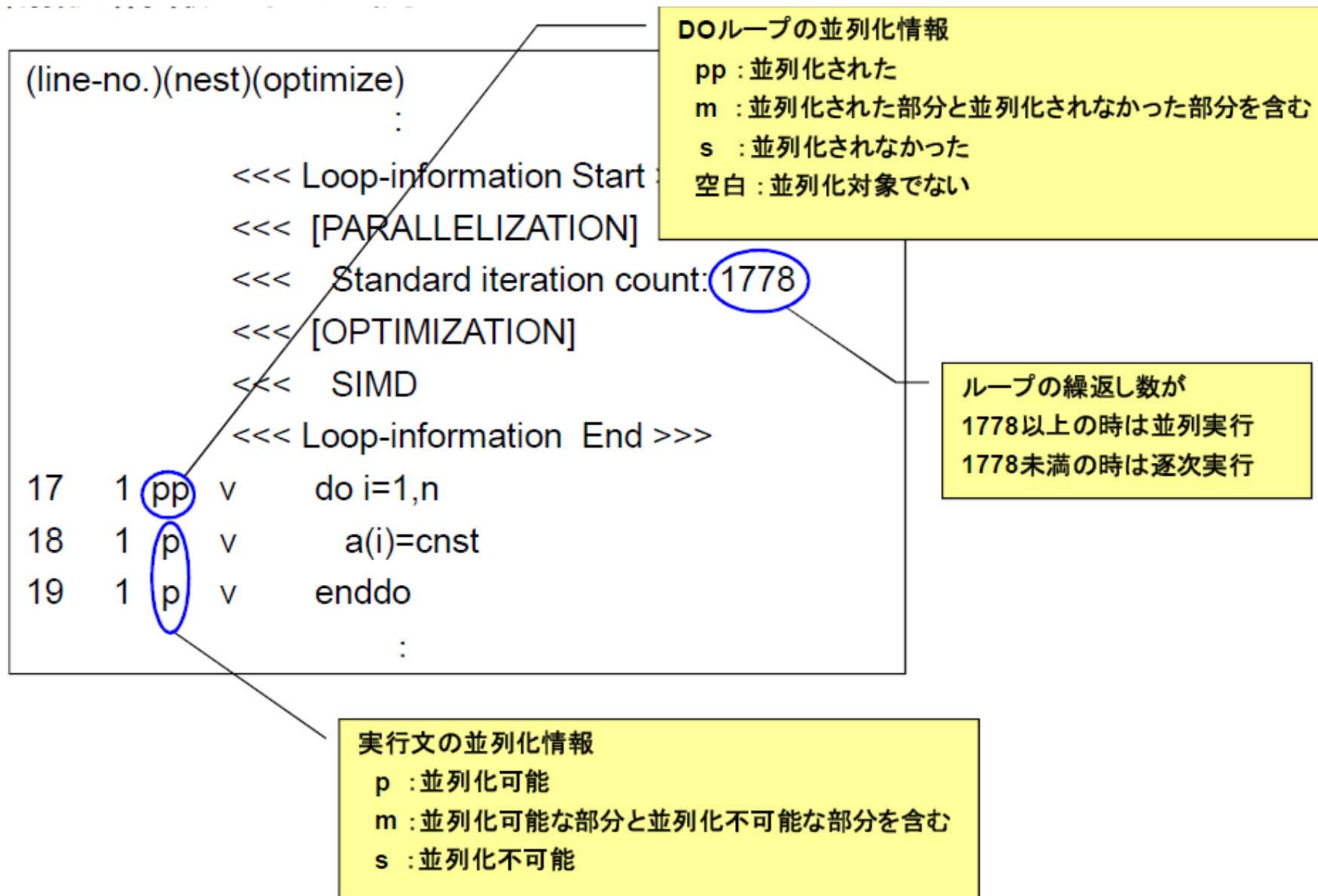
＊.lstの見方



SIMD情報



自動並列化情報



solver_ICCG_mc.lst (src)

```
      101      1                 !C
      102      1                 !C +----------------+
      103      1                 !C | {z}= [Minv]{r} |
      104      1                 !C +----------------+
      105      1                 !C===
      106      1
      107      1                 !$omp parallel do private(ip,i)
      108      2   p                   do ip= 1, PEsmpTOT
                               <<< Loop-information Start >>>
                               <<<  [OPTIMIZATION]
                               <<<    SIMD
                               <<<    SOFTWARE PIPELINING
                               <<< Loop-information  End >>>
      109      3   p   8v              do i = SMPindexG(ip-1)+1, SMPindexG(ip)
      110      3   p   8v                W(i,Z)= W(i,R)
      111      3   p   8v              enddo
      112      2   p                   enddo
      113      1                 !$omp end parallel do
      114      1
      115      1                       Stime= omp_get_wtime()
      116      1                       call fapp_start ("precond", 1, 1)
      117      2                       do ic= 1, NCOLORtot
      118      2                 !$omp parallel do private(ip,ip1,i,WVAL,k)
      119      3   p                     do ip= 1, PEsmpTOT
      120      3   p                       ip1= (ic-1)*PEsmpTOT + ip
      121      4   p                     do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      122      4   p                       WVAL= W(i,Z)
                               <<< Loop-information Start >>>
                               <<<  [OPTIMIZATION]
                               <<<    SIMD
                               <<<    SOFTWARE PIPELINING
                               <<< Loop-information  End >>>
      123      5   p   4v                  do k= indexL(i-1)+1, indexL(i)
      124      5   p   4v                    WVAL= WVAL -  AL(k) * W(itemL(k),Z)
      125      5   p   4v                  enddo
      126      4   p                       W(i,Z)= WVAL * W(i,DD)
      127      4   p                     enddo
      128      3   p                     enddo
      129      2                 !$omp end parallel do
      130      2                       enddo
```

solver_ICCG_mc.lst (short)

SOFTWARE PIPELININGが無い

```
   101      1                !C
   102      1                !C +-----------------+
   103      1                !C | {z}= [Minv]{r} |
   104      1                !C +-----------------+
   105      1                !C===
   106      1
   107      1                !$omp parallel do private(ip,i)
   108      2   p                  do ip= 1, PEsmpTOT
                           <<< Loop-information Start >>>
                           <<<  [OPTIMIZATION]
                           <<<    SIMD
                           <<< Loop-information  End >>>
   109      3   p    v             do i = SMPindexG(ip-1)+1, SMPindexG(ip)
   110      3   p    v               W(i,Z)= W(i,R)
   111      3   p    v             enddo
   112      2   p                  enddo
   113      1                !$omp end parallel do
   114      1
   115      1                      Stime= omp_get_wtime()
   116      1                      call fapp_start ("precond", 1, 1)
   117      2       3              do ic= 1, NCOLORtot
   118      2       3        !$omp parallel do private(ip,ip1,i,WVAL,k)
   119      3   p   3                do ip= 1, PEsmpTOT
   120      3   p   3                  ip1= (ic-1)*PEsmpTOT + ip
   121      4   p   3                do i= SMPindex(ip1-1)+1, SMPindex(ip1)
   122      4   p   3                  WVAL= W(i,Z)
                           <<< Loop-information Start >>>
                           <<<  [OPTIMIZATION]
                           <<<    SIMD
                           <<< Loop-information  End >>>
   123      5   p   3v                 do k= indexL(i-1)+1, indexL(i)
   124      5   p   3v                   WVAL= WVAL -  AL(k) * W(itemL(k),Z)
   125      5   p   3v                 enddo
   126      4   p   3                  W(i,Z)= WVAL * W(i,DD)
   127      4   p   3                enddo
   128      3   p   3                enddo
   129      2       3        !$omp end parallel do
   130      2       3              enddo
```

実習(3)

- Kshortloop=Nの値を変えて見よ
- 色数によって効き具合がどう変わるか?

現　状

- コンパイラの成熟度に問題アリ
  - FORTRANはそれでもかなり良い
  - C，C++
- NUMA向け最適化
  - 京，FX10には効かない
  - とは言え，やらなくて良いわけではない
    - 将来のアーキテクチャ

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

```
!$omp parallel do private (i,VAL,j)
      do i= 1, N
        VAL= D(i)*W(i,P)
        do j= 1, 6
          VAL= VAL + AL(j,i)*W(IAL(j,i),P)
        enddo
        W(i,Q)= VAL
      enddo
```

N= 64^3: 2.99 GFLOPS
N= 100^3: 5.57
N= 128^3: 3.57

-Kfast, openmp

N= 64^3: 4.89 GFLOPS
N= 100^3: 7.36
N= 128^3: 6.70