Introduction to Parallel Programming for Multicore/Manycore Clusters

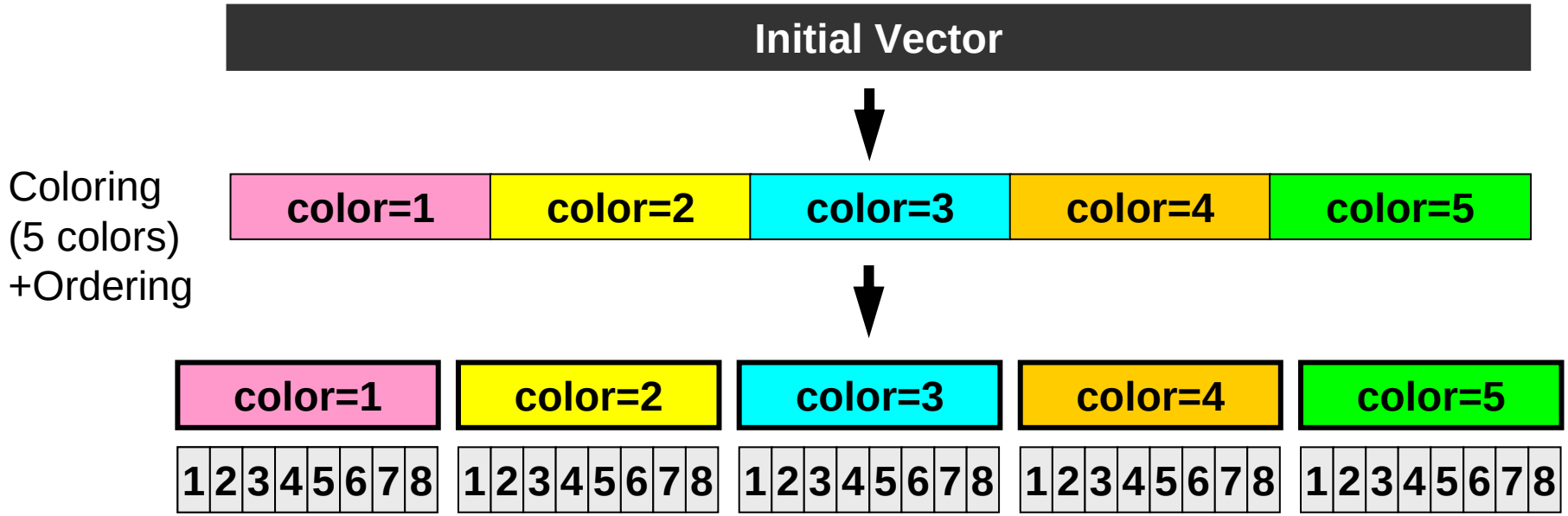
Part VI: Report

Kengo Nakajima
Information Technology Center

SMPindex:

for preconditioning

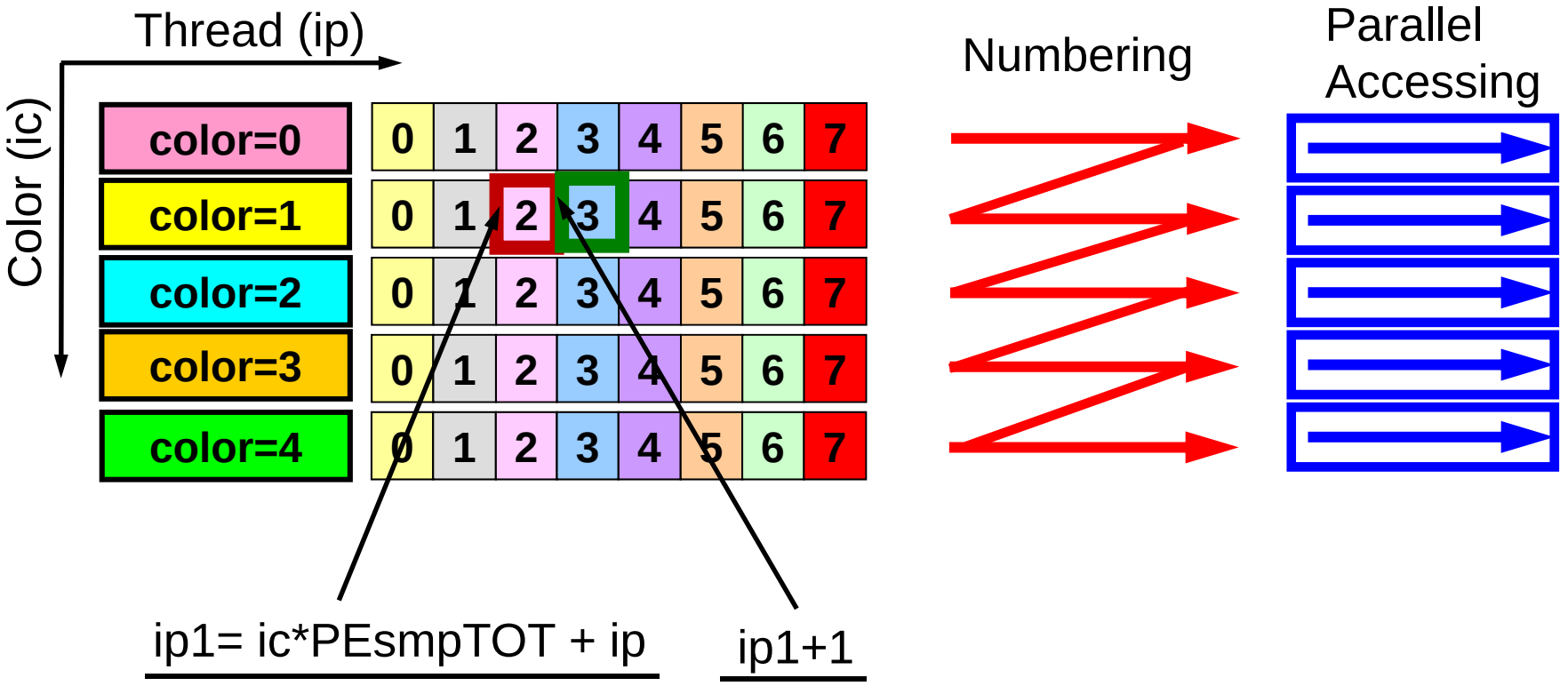
```
for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for ...
    for(ip=0; ip<PEsmpTOT; ip++) {
        ip1 = ic * PEsmpTOT + ip;
        for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {
            (...)
        }
    }
}
```



- 5-colors, 8-threads
- Meshes in same color are independent: parallel processing
- Reordering in ascending order according to color ID

SMPindex

```
#pragma omp parallel private (ic,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++) {…
```



SMPindex

```
COLORindex[ic  ]= 100
COLORindex[ic+1]= 200
PEsmpTOT        =   8

nn1= 200-100    = 100
num= 100 / 8    =  12 (12.5)
nr = 100 - 12*8=   4

SMPindex[ip1  ]= 100
SMPindex[ip1+1]= 113 (13 elements in the 1st thread)
SMPindex[ip1+2]= 126 (13 elements in the 2nd thread)
SMPindex[ip1+3]= 139 (13 elements in the 3rd thread)
SMPindex[ip1+4]= 152 (13 elements in the 4th thread)
SMPindex[ip1+5]= 164 (12 elements in the 5th thread)
SMPindex[ip1+6]= 176 (12 elements in the 6th thread)
SMPindex[ip1+7]= 188 (12 elements in the 7th thread)
SMPindex[ip1+8]= 200 (12 elements in the 8th thread)
```

Problems in Reordering

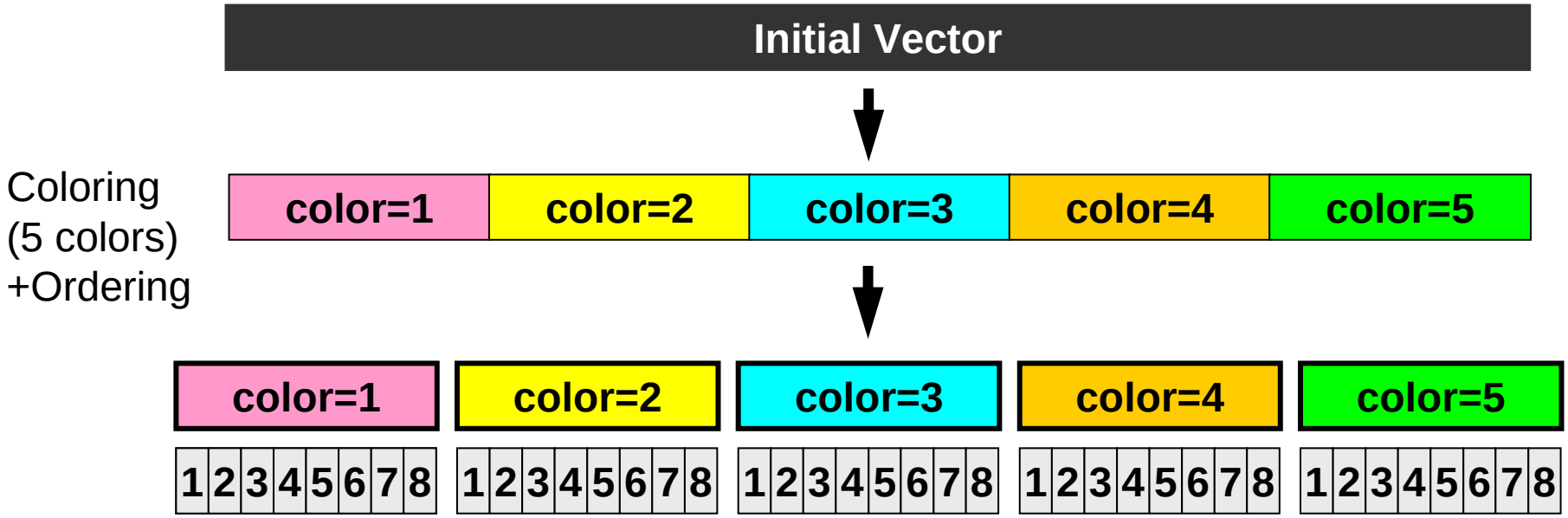
- Coloring
  - MC
  - RCM
  - CM-RCM
- Renumbering is according to color/level ID

- On each thread, numbering is not continuous
  - reduced performance

SMPindex:

for preconditioning

```
for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for ...
    for(ip=0; ip<PEsmpTOT; ip++) {
        ip1 = ic * PEsmpTOT + ip;
        for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {
            (...)
        }
    }
}
```



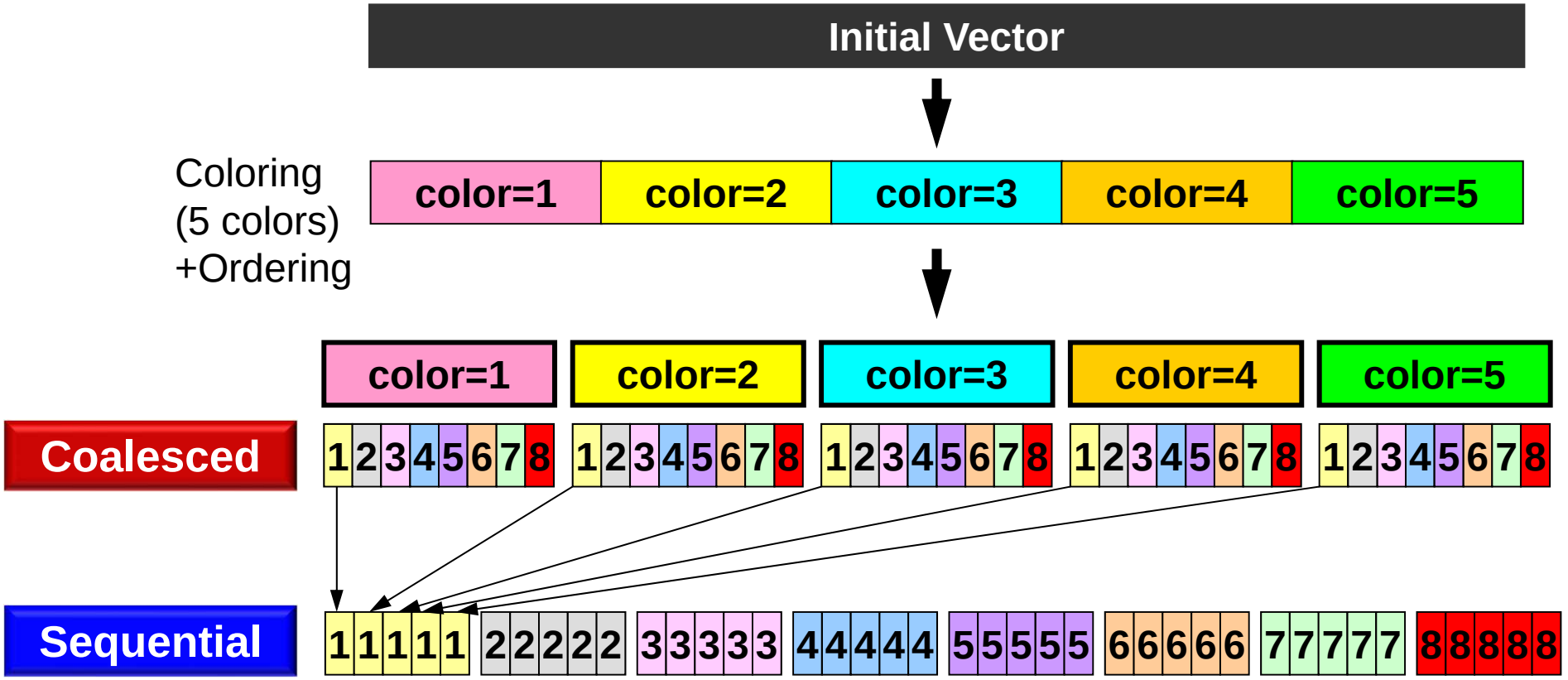
- 5-colors, 8-threads
- Meshes in same color are independent: parallel processing
- Reordering in ascending order according to color ID

Sequential Reordering

- Reordering for continuous memory access on each thread (core)
  - Performance is expected to be better.
    - Continuous address of arrays, such as coefficient matrices
    - Locality (2-page later)

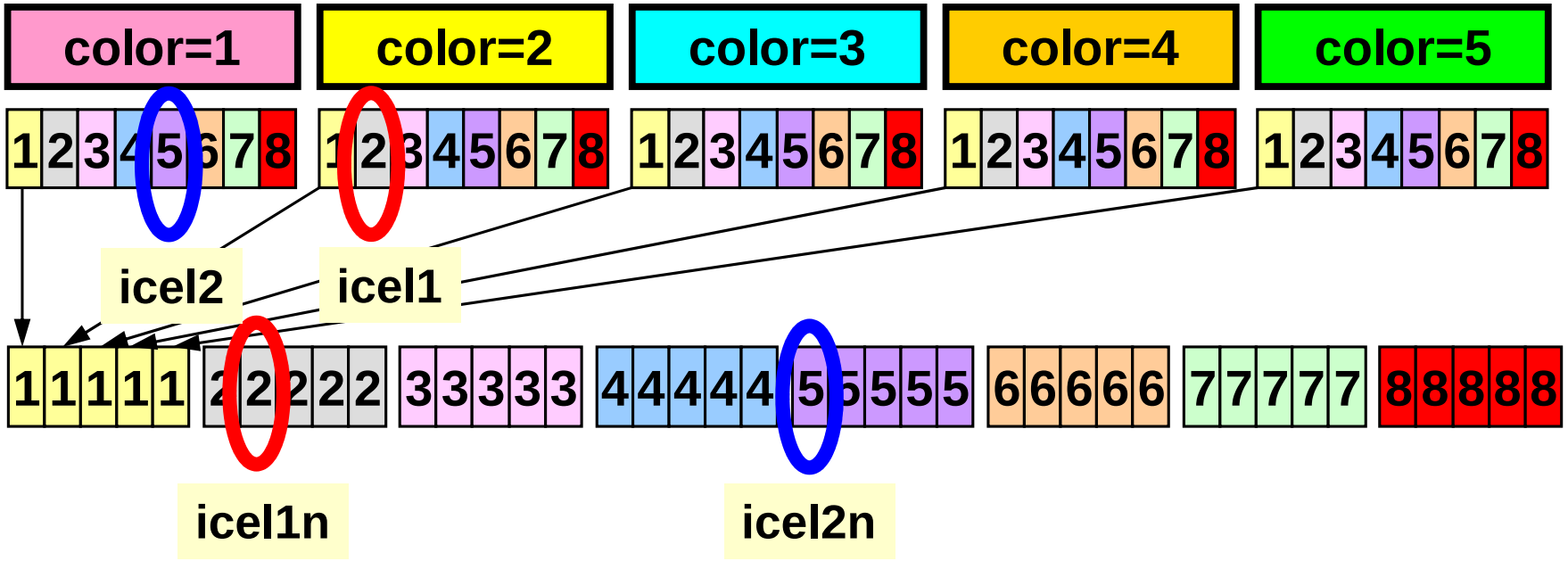
Sequential Reordering

Further reordering for continuous memory access on each thread, 5-color, 8-threads



Inconsistent numbering may occur

- Coalesced
  - **`icel1 > icel2`**, therefore, **`icel2= itemL[k]`**, where **`indexL[icel1]`**$\leqq$**`k`**$<$**`indexL[icel1+1]`**
- Sequential
  - **`icel1n < icel2n`**, but still **`icel2n= itemL[k]`**, where **`indexL[icel1n]`**$\leqq$**`k`**$<$**`indexL[icel1n+2]`**



Sequential Reordering

CM-RCM(2), 4-threads

Continuous Data Access on a Thread: Utilization of Cache, Prefetching

45	10	39	5	35	2	33	1
17	46	11	40	6	36	3	34
53	18	47	12	41	7	37	4
24	54	19	48	13	42	8	38
59	25	55	20	49	14	43	9
29	60	26	56	21	50	15	44
63	30	61	27	57	22	51	16
32	64	31	62	28	58	23	52

CM-RCM(2)

29	18	15	5	11	2	9	1
33	30	19	16	6	12	3	10
45	34	31	20	25	7	13	4
40	46	35	32	21	26	8	14
59	49	47	36	41	22	27	17
53	60	50	48	37	42	23	28
63	54	61	51	57	38	43	24
56	64	55	62	52	58	39	44

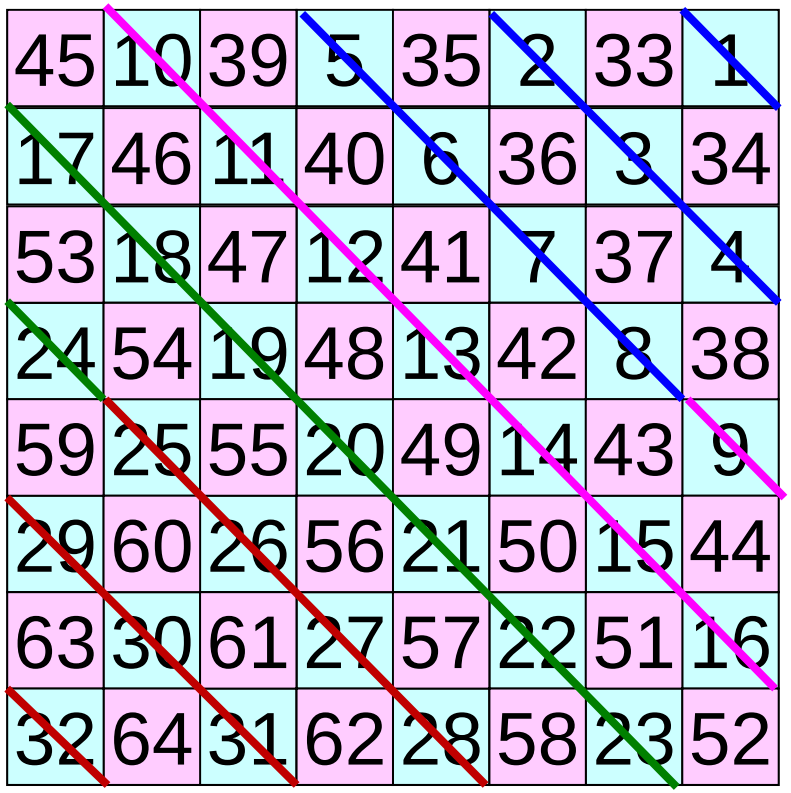
Sequential Reordering, 4-threads

Sequential Reordering

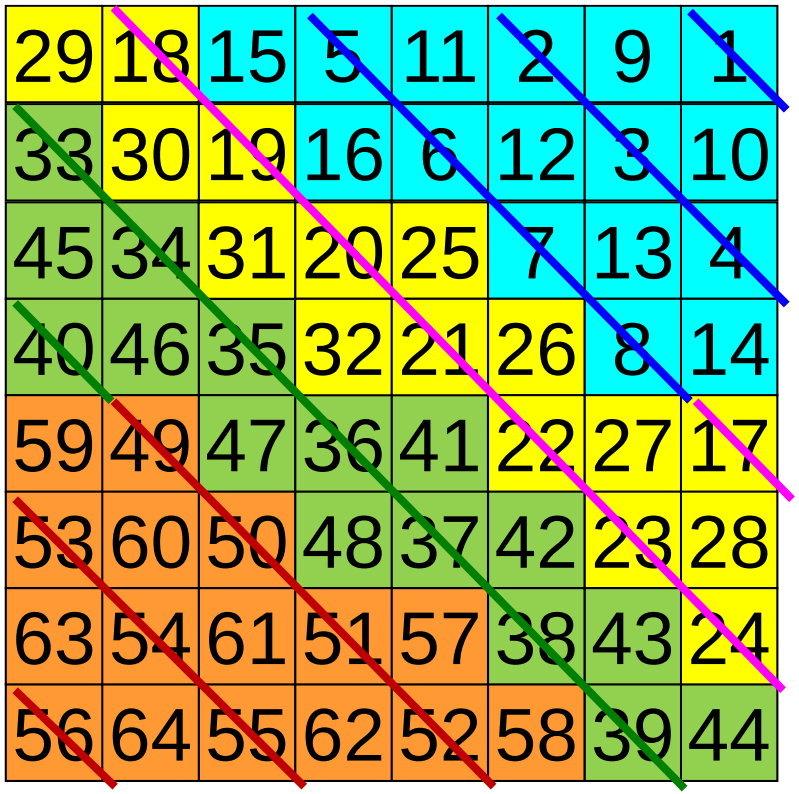
CM-RCM(2), 4-threads

1st-Color

#0 thread, #1, #2, #3



CM-RCM(2)



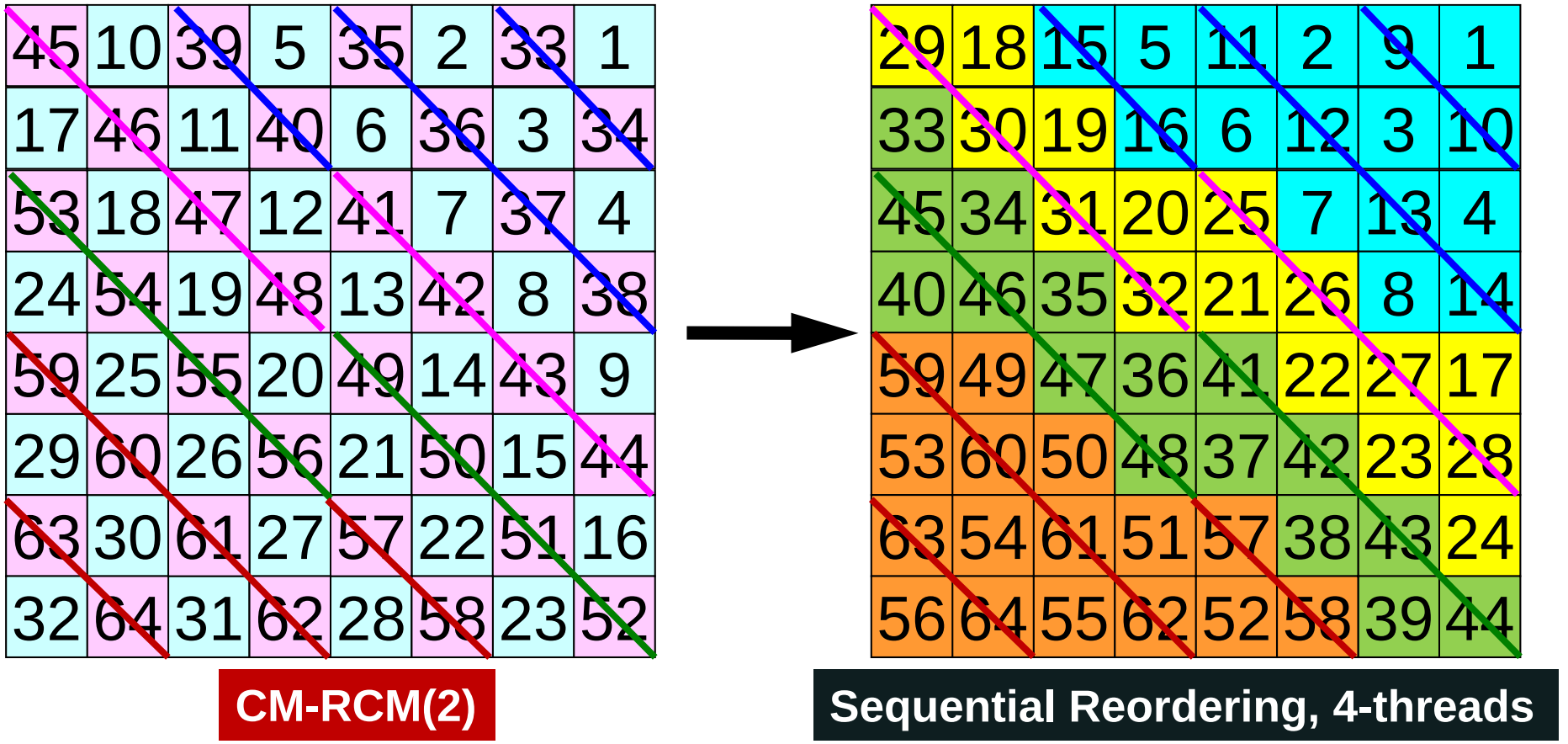
Sequential Reordering, 4-threads

Sequential Reordering

CM-RCM(2), 4-threads

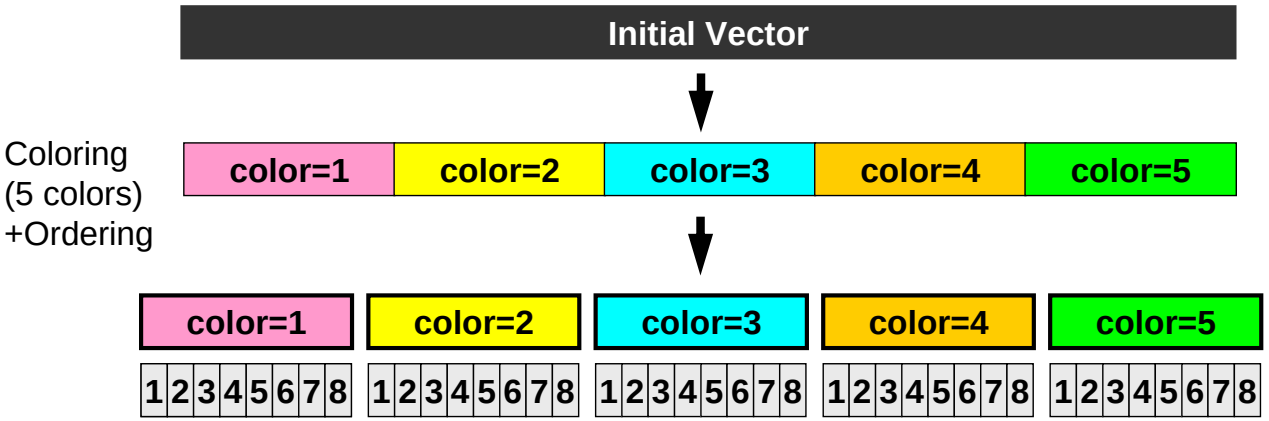
2nd-Color

#0 thread, #1, #2, #3

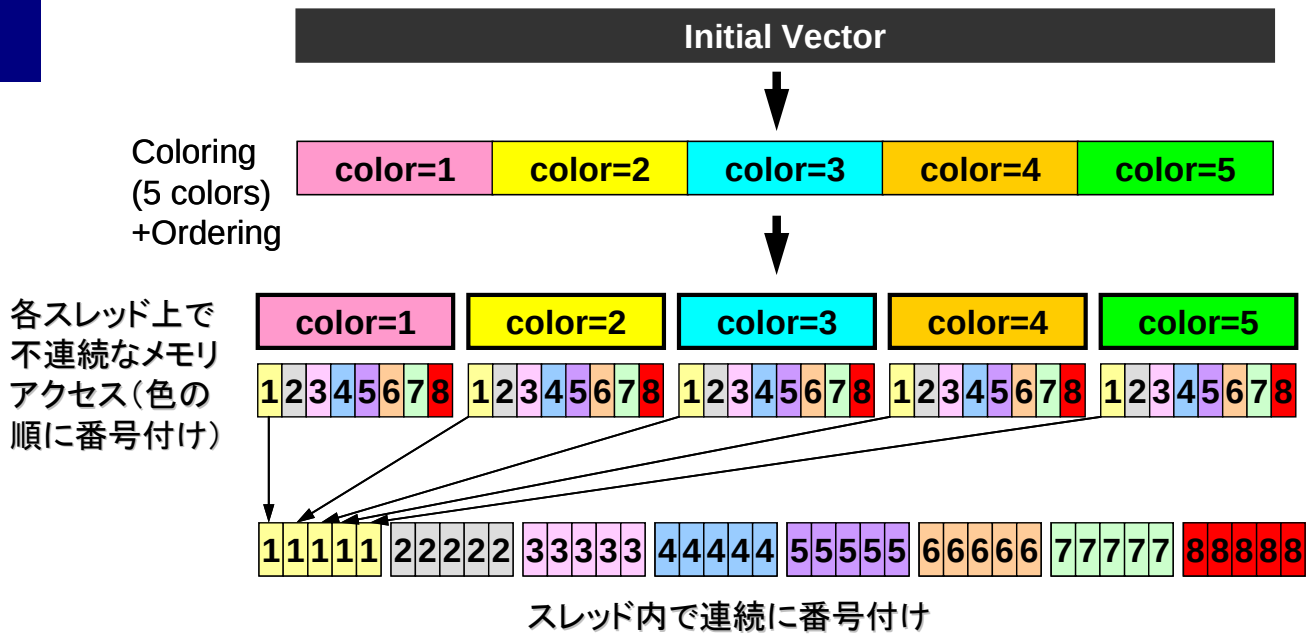


Sequential Reordering

Coalesced
Good for GPU



Sequential



Additional Code

- Location
  - **<$O-L3>: /lustre/gt12/tXXYYY/multicoe/L3**
  - **<$O-L3>/reorder0, <$O-L3>/run**
- Compile & Run
  - Source Code
    - `cd <$O-L3>/reorder0`
    - `make`
    - `<$O-L3>/run/L3-rsol0` execution file
  - Control Data
    - `<$O-L3>/run/INPUT.DAT`
  - Shell Script
    - `<$O-L3>/run/gor.sh`

gor.sh

```
#PBS -q u-lecture2
#PBS -N test
#PBS -l select=1:ncpus=16       ncpus=PEsmpTOT
#PBS -Wgroup_list=gt12
#PBS -l walltime=00:10:00
#PBS -e test2.err
#PBS -o test2.lst

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

export OMP_NUM_THREADS=16       =PEsmpTOT
export KMP_AFFINITY=granularity=fine,compact
numactl ./L3-rsol0
numactl ./L3-rsol0
numactl ./L3-rsol0
```

INPUT.DAT

```
100 100 100                          NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00           DX/DY/DZ
1.0e-08                              EPSICC
16                                   PEsmpTOT
-100                                     NCOLORtot
0                                    NFLAG
0                                    METHOD
```

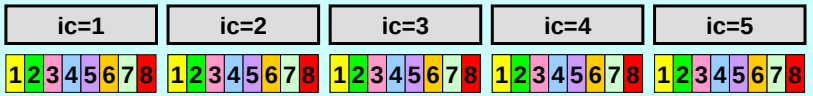
- **PEsmpTOT**
  - Thread Number **(= OMP_NUM_THREADS)**
- **NCOLORtot**
  - Reordering Method + Initial Number of Colors/Levels
  - ≧2: MC, =0: CM, =-1: RCM, -2≧: CMRCM
- **NFLAG**
  - =0: without first-touch, =1: with first-touch
- **METHOD**
  - Loop structure for Mat-Vec
  - =0: conventional way, =1: similar to forward/backward substitution

Sequential Reordering (1/4) Main

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "struct.h"
#include "pcg.h"
(…)

int main(){
        double *WK;
        int ISET, ITR, IER;
        int icel, ic0, i;
        double Stime, Etime;

        if(INPUT()) goto error;
        if(POINTER_INIT()) goto error;
        if(BOUNDARY_CELL()) goto error;
        if(CELL_METRICS()) goto error;
        if(POI_GEN()) goto error;

        ISET = 0;
        if(METHOD == 0 ){
                if(solve_ICCG_mc(ICELTOT, NL, NU, indexLnew, itemLnew,
                        indexUnew, itemUnew, D, BFORCE, PHI, ALnew, AUnew,
                        NCOLORtot, PEsmpTOT, SMPindex_new, EPSICCG,
                        &ITR, &IER)) goto error;
        }else if (METHOD == 1){
                if(solve_ICCG_mc_ft(ICELTOT, NL, NU, indexLnew, itemLnew,
                        indexUnew, itemUnew, D, BFORCE, PHI, ALnew, AUnew,
                        NCOLORtot, PEsmpTOT, SMPindex_new, EPSICCG,
                        &ITR, &IER)) goto error;
        }

        for(ic0=0; ic0<ICELTOT; ic0++){
                icel = NEWtoOLDnew[ic0];
                WK[icel-1] = PHI[ic0];
        }
        for(icel=0; icel<ICELTOT; icel++){
                PHI[icel] = WK[icel];
        }

        if(OUTUCD()) goto error;
        return 0;

error:
        return -1;
}
```

```
SMPindex = (int *) allocate_vector(sizeof(int), NCOLORtot * PEsmpTOT + 1);
memset(SMPindex, 0, sizeof(int)*(NCOLORtot*PEsmpTOT+1));

for(ic=1; ic<=NCOLORtot; ic++) {
    nn1 = COLORindex[ic] - COLORindex[ic-1];
    num = nn1 / PEsmpTOT;
    nr  = nn1 - PEsmpTOT * num;
    for(ip=1; ip<=PEsmpTOT; ip++) {
        if(ip <= nr) {
           SMPindex[(ic-1)*PEsmpTOT+ip] = num + 1;
        } else {
           SMPindex[(ic-1)*PEsmpTOT+ip] = num;
        }
    }
}
```

SMPindex



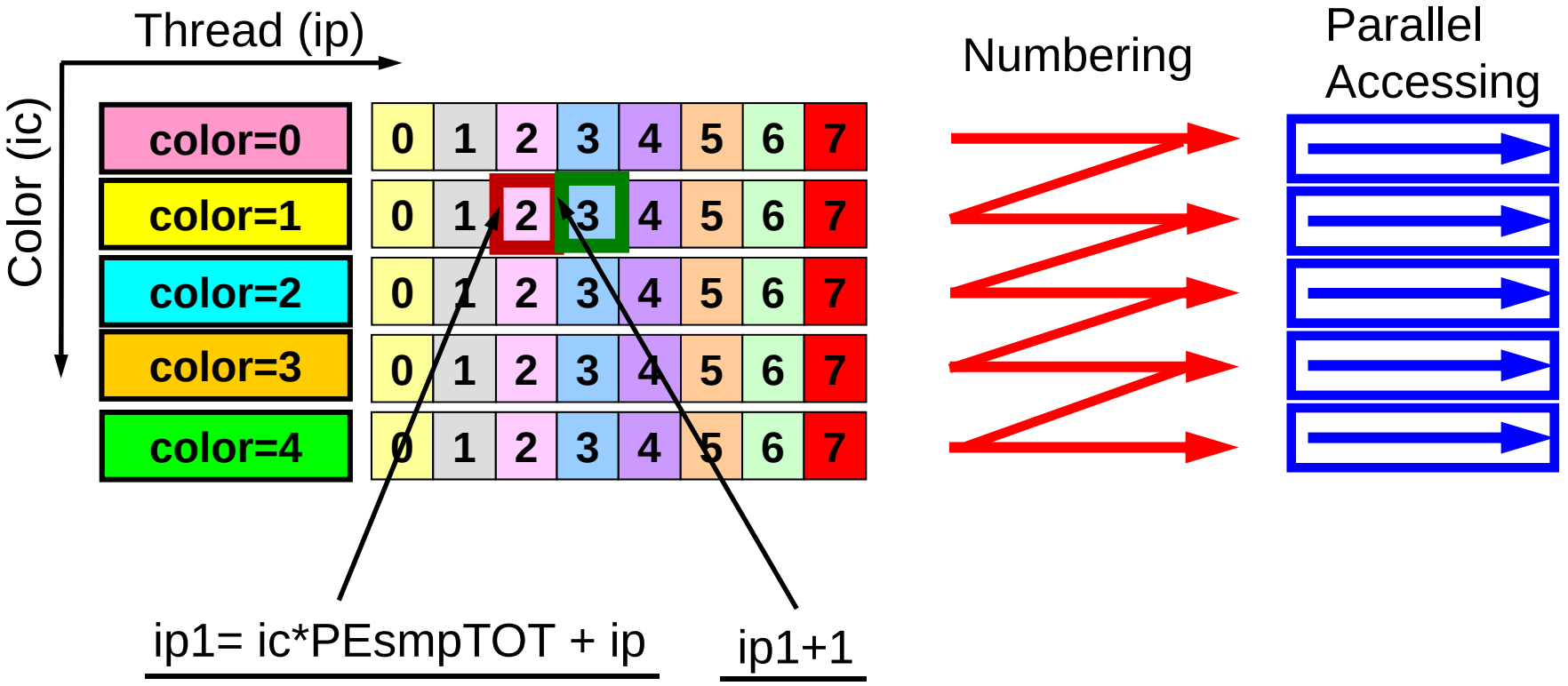
SMPindex_new


```
SMPindex_new = (int *) allocate_vector(sizeof(int), NCOLORtot * PEsmpTOT + 1);
memset(SMPindex_new, 0, sizeof(int)*(NCOLORtot*PEsmpTOT+1));

for(ic=1; ic<=NCOLORtot; ic++) {
    for(ip=1; ip<=PEsmpTOT; ip++) {
        j1 =(ic-1)*PEsmpTOT + ip;
        j0 =j1-1;
        SMPindex_new[(ip-1)*NCOLORtot+ic] = SMPindex[j1];
        SMPindex[j1] = SMPindex[j0] + SMPindex[j1];
    }
}

for(ip=1; ip<=PEsmpTOT; ip++) {
    for(ic=1; ic<=NCOLORtot; ic++) {
        j1 = (ip-1) * NCOLORtot + ic;
        j0 = j1 - 1;
        SMPindex_new[j1] += SMPindex_new[j0];
    }
}
```

Sequential Reordering (2/4) poi_gen-1

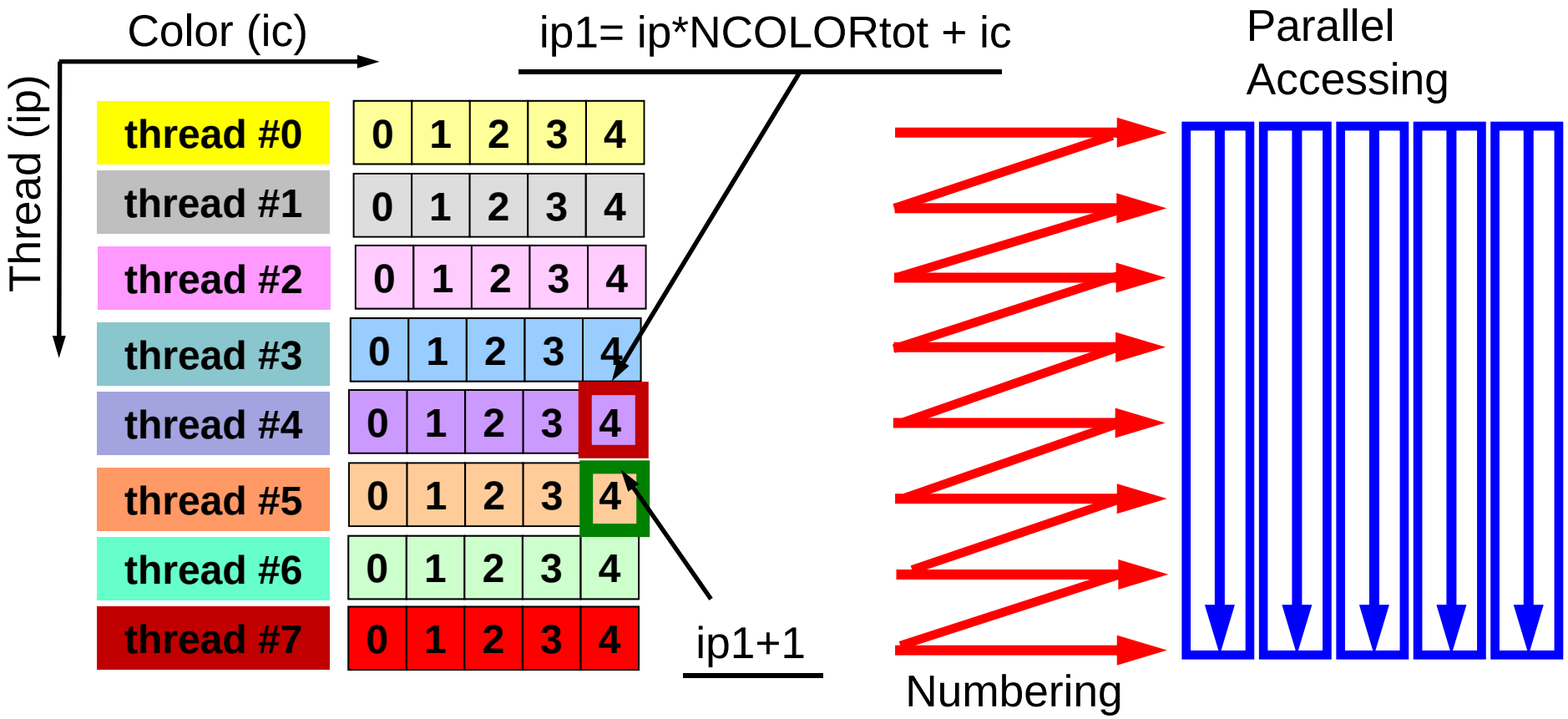
Coalesced

```
#pragma omp parallel private (ic,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++) {…
```



Sequential

```
#pragma omp parallel private (ic,ip,ip1,i,WVAL,j)
for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp for
   for(ip=0; ip<PEsmpTOT; ip++) {
       ip1 = ip * NCOLORtot + ic;
       for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {…
```



Sequential Reordering (3/4) poi_gen-2

```
for(ip=0; ip<PEsmpTOT; ip++){
    for(ic=0; ic<NCOLORtot; ic++){
        icNS = SMPindex_new[ip*NCOLORtot + ic];
        ic01 = SMPindex[ic*PEsmpTOT + ip];
        ic02 = SMPindex[ic*PEsmpTOT + ip+1];
        icou = 0;
        for(k=ic01; k<ic02; k++){
            icel=NEWtoOLD[k];
            icou = icou +1;
            icelN=icNS+icou;
            OLDtoNEWnew[icel-1] = icelN;
            NEWtoOLDnew[icelN-1]= icel;
        }
    }
}
```

```
for(ip=1; ip<=PEsmpTOT; ip++) {
    id1 = ip    *NCOLORtot;
    id2 = (ip-1)*NCOLORtot;

    for(icel=SMPindex_new[id2]+1; icel<=SMPindex_new[id1]; icel++) {
        ic0  = NEWtoOLDnew[icel-1];
        ik0  = OLDtoNEW[ic0-1];

        INLnew[icel-1] =INL[ik0-1];
        INUnew[icel-1] =INU[ik0-1];
    }
}


for(i=1; i<=ICELTOT; i++){
    indexL[i]=indexL[i-1]+INLnew[i-1];
    indexU[i]=indexU[i-1]+INUnew[i-1];
}
```

OLDtoNEWnew: Original -> Sequential
NEWtoOLDnew: Sequential -> Original

-Original: Initial
-Coalesced
-Sequential

```
#pragma omp parallel for private (icel,id1, id2 …)
for(ip=1; ip<=PEsmpTOT; ip++) {
    id1= ip    *NCOLORtot; id2= (ip-1)*NCOLORtot;
    for(icel=SMPindex_new[id2]+1; icel<=SMPindex_new[id1]; icel++) {
        ic0  = NEWtoOLDnew[icel-1];
        ik0  = OLDtoNEW[ic0-1];
        icN10 = NEIBcell[ic0-1][0];
(…)
        icN50 = NEIBcell[ic0-1][4];
        icN60 = NEIBcell[ic0-1][5];
        isL=indexL[ik0-1]; ieL=indexL[ik0  ];
        isU=indexU[ik0-1]; ieU=indexU[ik0  ];

        if(icN50 != 0) {
           icN5 = OLDtoNEW[icN50-1];
           coef = RDZ * ZAREA;
           D[icel-1] -= coef;
           if(icN5 < ik0) {
              for(j=isL; j<ieL; j++) {
                  if(itemL[j] == icN5) {
                     j_new=indexLnew[icel-1]+j-isL;
                     ALnew[j_new]    = coef;
                     itemLnew[j_new] = OLDtoNEWnew[icN50-1];
                     break;
                  }
              }
           } else {
              for(j=isU; j<ieU; j++) {
                  if(itemU[j] == icN5) {
                     j_new=indexUnew[icel-1]+j-isU;
                     AUnew[j_new] = coef;
                     itemUnew[j_new] = OLDtoNEWnew[icN50-1];
                     break;
                  }
              }
           }
    }
```

Sequential Reordering (4/4) poi_gen-3

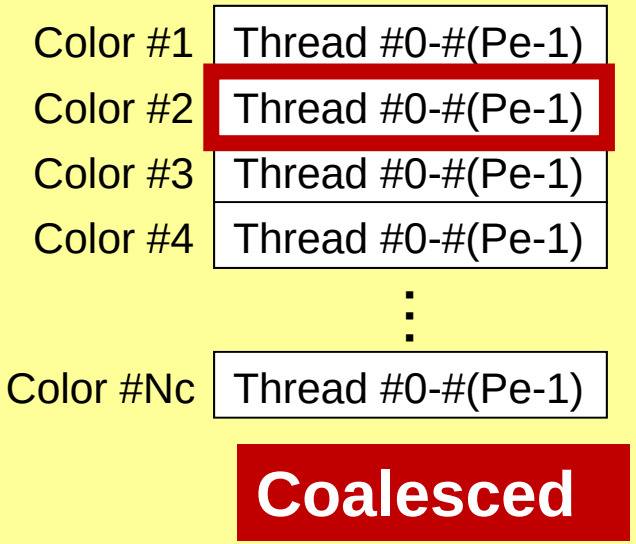
icel: Sequential
ic0: Original
ik0: Coalesced

icN50: Original
icN5 : Coalesced

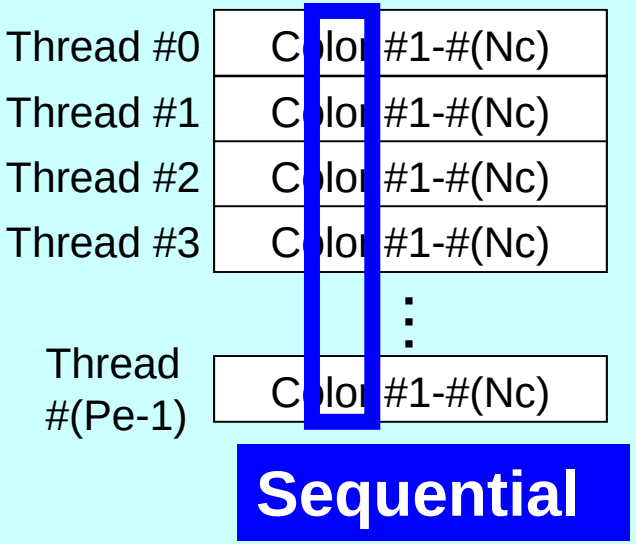
icN5>ik0: Upper (AU)
icN5<ik0: Lower (AL)

Forward Substitution

```
#pragma omp parallel private (ic,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];
        }
    }
}
```




```
#pragma omp parallel private (ic,ip,ip1,i,WVAL,j)
for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp for
    for(ip=0; ip<PEsmpTOT; ip++) {
        ip1 = ip * NCOLORtot + ic;
        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];
        }
    }
}
```



Mat-Vec

```
#pragma omp parallel for private(ip,i)
for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
        VAL = D[i] * W[P][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            VAL += AL[j] * W[P][itemL[j]-1];
        }
        for(j=indexU[i]; j<indexU[i+1]; j++) {
            VAL += AU[j] * W[P][itemU[j]-1];
        }
        W[Q][i] = VAL;
    }
}
```

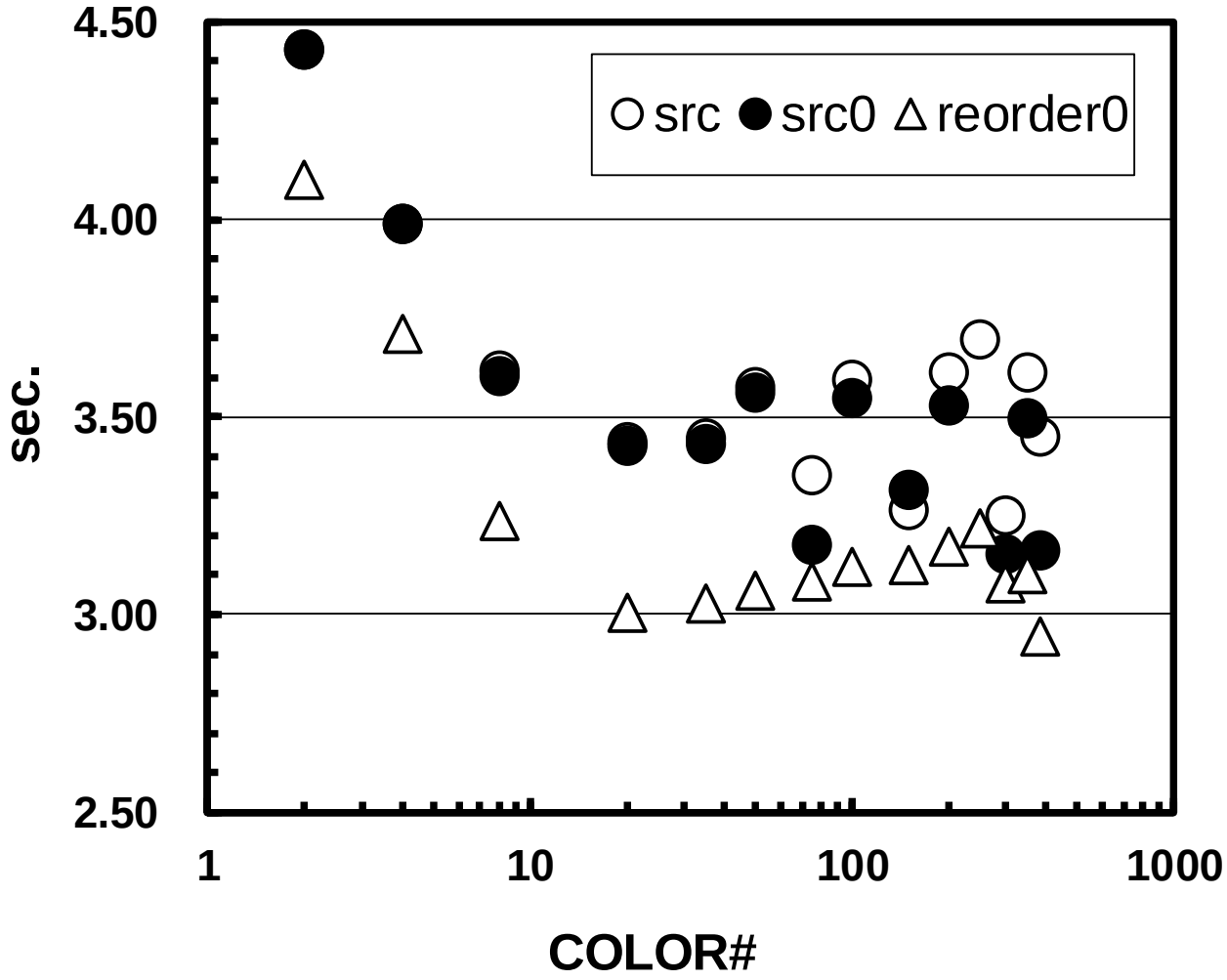
METHOD=0

```
#pragma omp parallel for private (ip1,i,VAL,j)
for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindex[ip*NCOLORtot]; i<SMPindex[(ip+1)*NCOLORtot];i++){
        VAL = D[i] * W[P][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            VAL += AL[j] * W[P][itemL[j]-1];
        }
        for(j=indexU[i]; j<indexU[i+1]; j++) {
            VAL += AU[j] * W[P][itemU[j]-1];
        }
        W[Q][i] = VAL;
    }
}
```

METHOD=1

Comp. Time for ICCG, CM-RCM

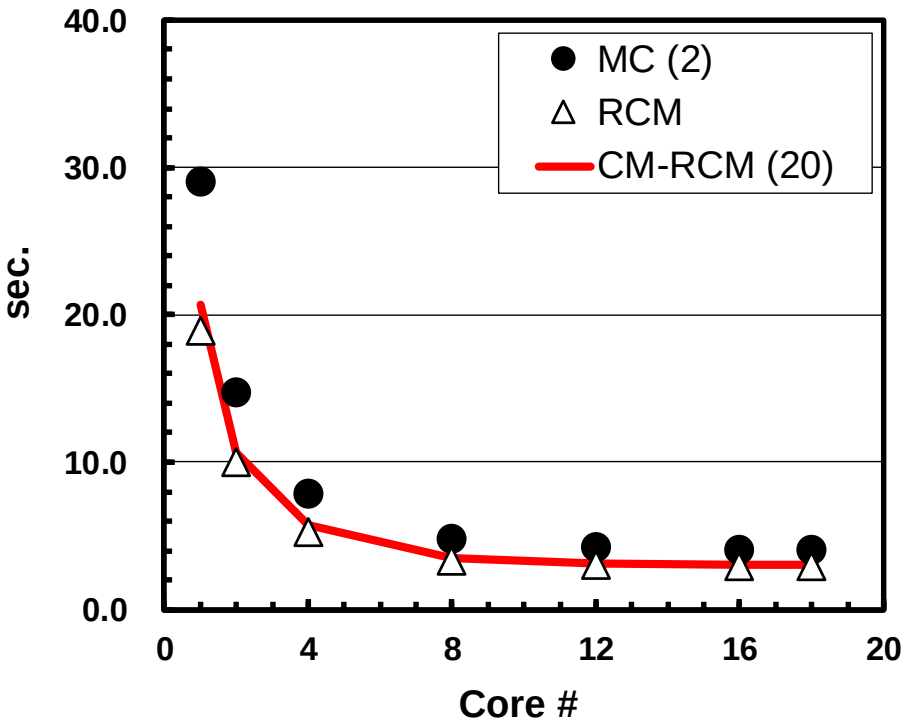
Generally “sequential (reorder0)” is stable and faster than “coalesced (src, src0)”. Effects should be more significant in cases with more colors, but … (16 threads)



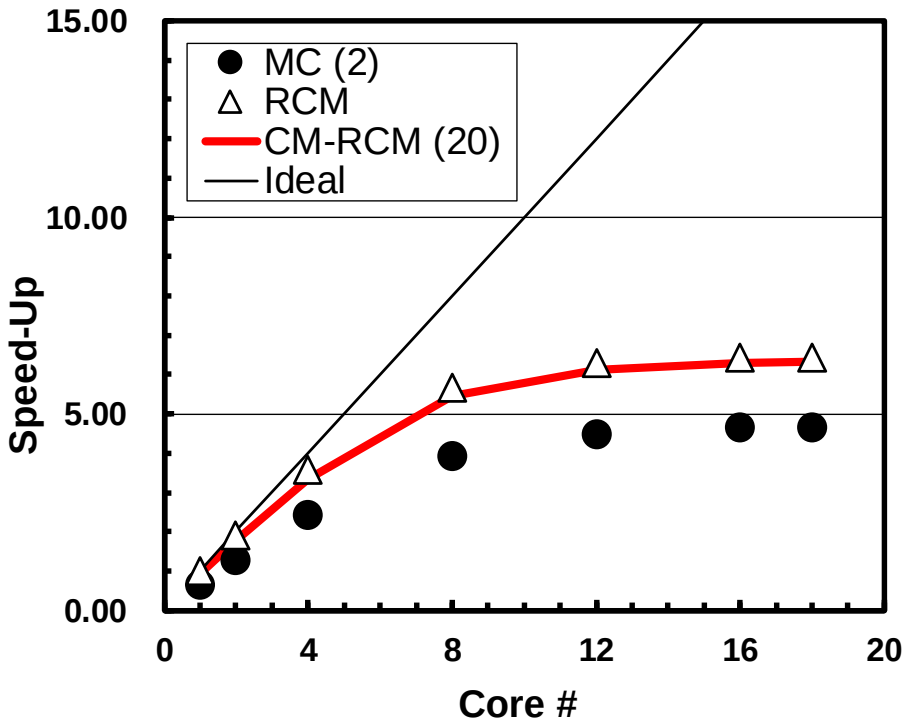
Results on Reedbush-U：128^3

18 cores, **reorder0**：MC（2-colors）：424 iter's, 4.05 sec.
RCM（382-levels）：287 iter's, 2.94 sec.
CM-RCM（Nc=20）：318 iter's, 2.99 sec.

Computation Time



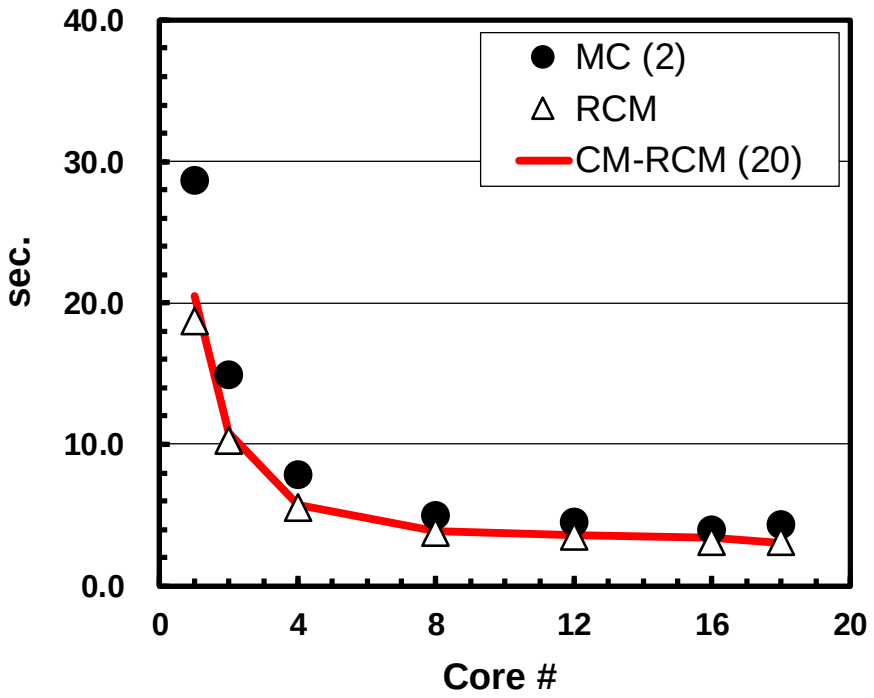
Speed-Up based on RCM with 1 thread



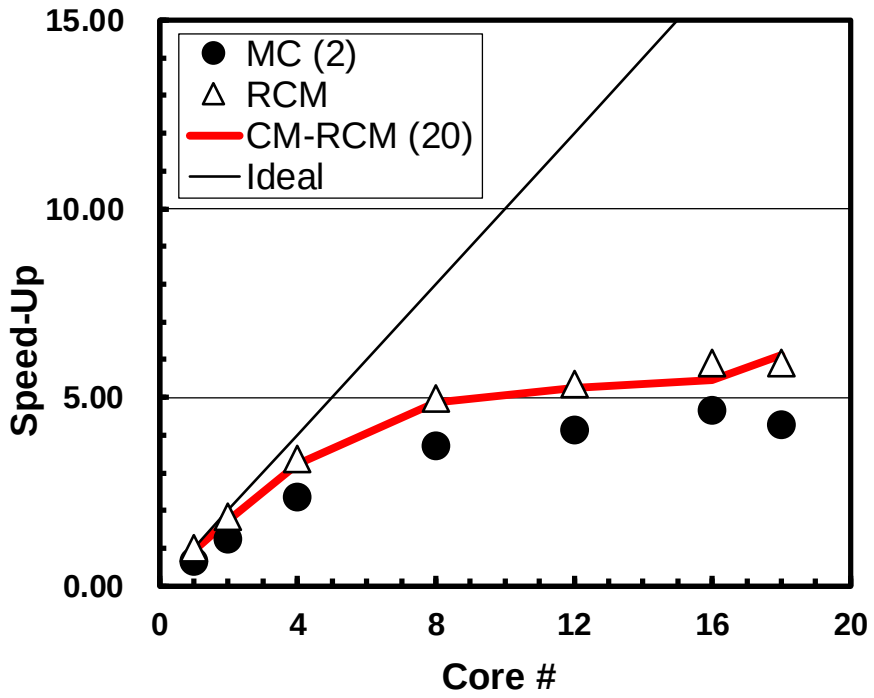
Results on Reedbush-U：128^3

18 cores, **src**: MC（2-colors）：424 iter's, 4.38 sec.
RCM（382-levels）：287 iter's, 3.16 sec.
CM-RCM（Nc=20）：318 iter's, 3.06 sec.

Computation Time

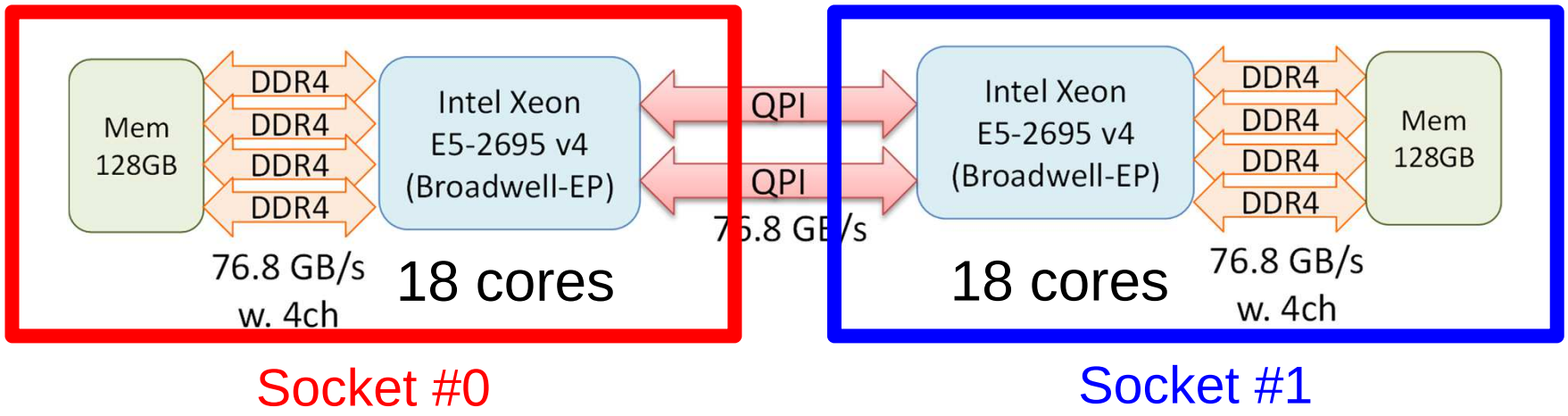


Speed-Up based on RCM with 1 thread



Reedbush-U

1-node: 2-CPU's/sockets



- Each Node of Reedbush-U
  - 2 Sockets (CPU's) of Intel Broadwell-EP
  - Each socket has 18 cores
- Each core of a socket can access to the memory on the other socket : NUMA (Non-Uniform Memory Access)
- Utilization of the local memory is more efficient
- So far, only a single socket has been used
  - Let's utilize both sockets

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.

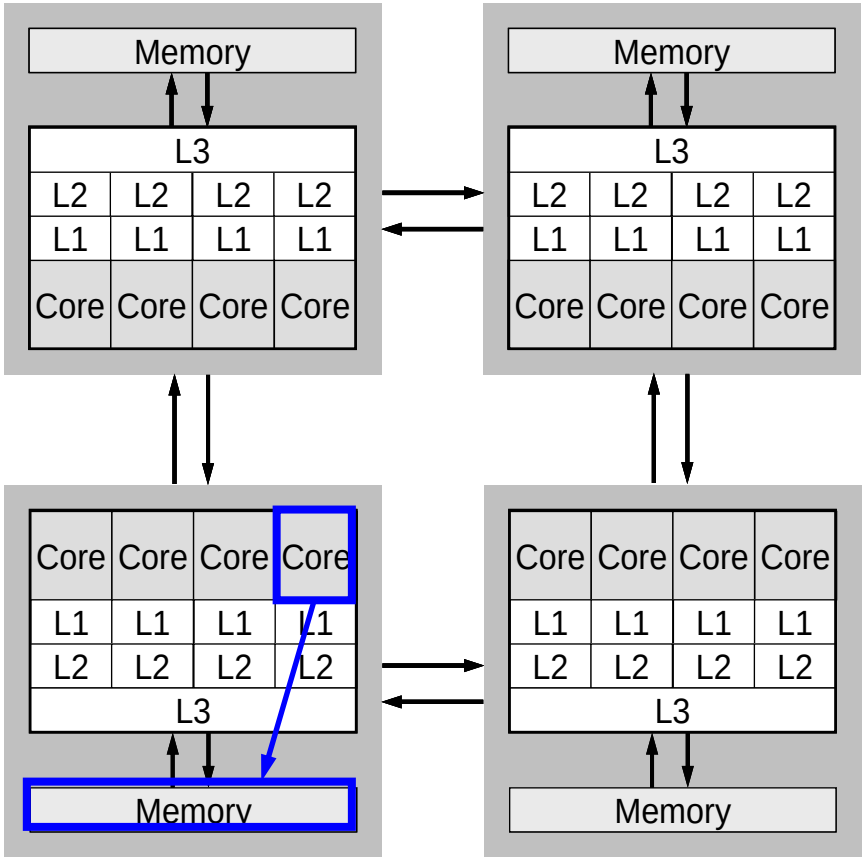
The most common NUMA page-placement algorithm is the “first touch” algorithm, in which the PE first referencing a region of memory will have the page holding that memory assigned to it.

A very common technique in OpenMP program for optimization is to initialize data in parallel using the same loop schedule as will be used later in the computations.

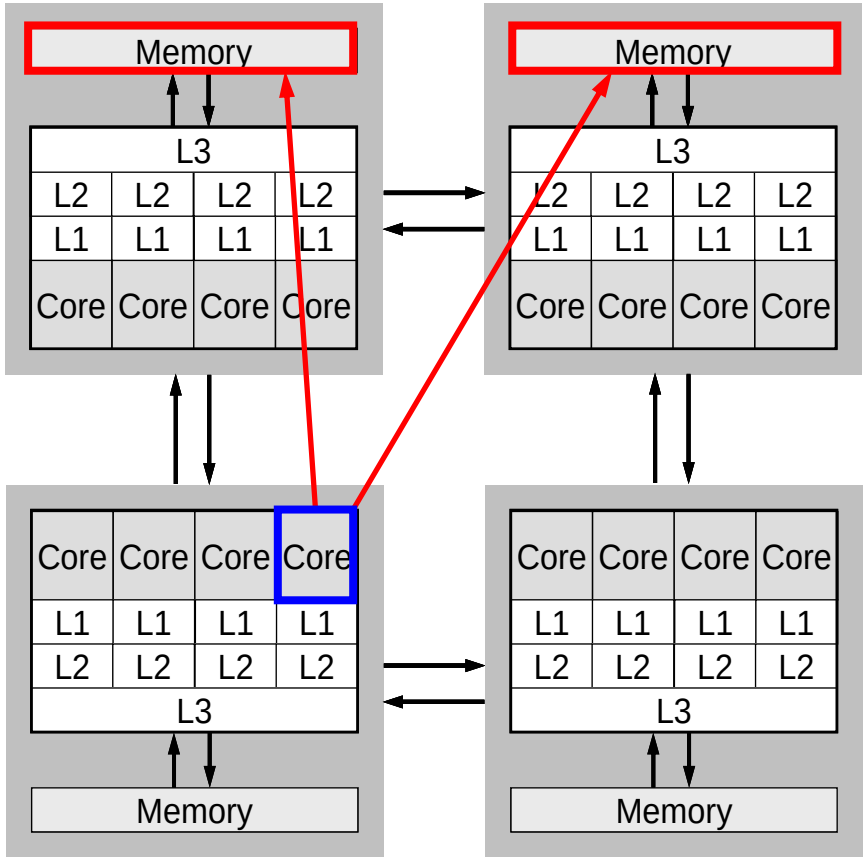
Summary: First Touch Data Placement

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

Local/Remote Memory



Local Memory



Remote Memory

Control Data: INPUT.DAT

```
128 128 128                    NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00     DX/DY/DZ
1.0e-08                        EPSICC
36                             PEsmpTOT
-50                            NCOLORtot
0                              NFLAG (0 or 1)
0                              METHOD
```

- **PEsmpTOT**
  - Thread Number **(= OMP_NUM_THREADS)**
- **NCOLORtot**
  - Reordering Method + Initial Number of Colors/Levels
  - ≧2: MC, =0: CM, =-1: RCM, -2≧: CMRCM
- **NFLAG**
  - =0: without first-touch, =1: with first-touch
- **METHOD**
  - Loop structure for Mat-Vec
  - =0: conventional way, =1: similar to forward/backward substitution

go2.sh: reorder0

```
#PBS -q u-lecture2
#PBS -N test
#PBS -l select=1:ncpus=36       ncpus=PEsmpTOT
#PBS -Wgroup_list=gt12
#PBS -l walltime=00:10:00
#PBS -e test2.err
#PBS -o test2.lst

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

export OMP_NUM_THREADS=36       =PEsmpTOT
export KMP_AFFINITY=granularity=fine,compact
numactl ./L3-rsol0
numactl ./L3-rsol0
numactl ./L3-rsol0
```

Array Initialization: NFLAG=0/1 (1/2)

```
if(NFLAG == 0){
   for(i=0; i<ICELTOT; i++) {
       BFORCE[i] = 0.0;
       D[i]      = 0.0;
       PHI[i]    = 0.0;
   }
   for(i=0; i<=ICELTOT; i++) {
       indexLnew[i] = indexLnew_org[i];
       indexUnew[i] = indexUnew_org[i];
   }
   for(i=0; i<NPL; i++) {
       itemLnew[i] = 0;
       ALnew[i] = 0.0;
   }
   for(i=0; i<NPU; i++) {
       itemUnew[i] = 0;
       AUnew[i] = 0.0;
   }

   }else {
```

Pages are allocated at the local memory of the master thread

A very common technique in OpenMP program for optimization is to initialize data in parallel using the same loop schedule as will be used later in the computations.

Array Initialization: NFLAG=0/1 (2/2)

```
    }else {
            indexLnew[0]=0;
            indexUnew[0]=0;
#pragma omp parallel for private (icel, j)
     for(ip=1; ip<=PEsmpTOT; ip++){
         for(icel = SMPindex_new[(ip-1)*NCOLORtot]+1;
             icel<=SMPindex_new[ip*NCOLORtot]; icel++){
             BFORCE[icel-1] = 0.0;
             PHI[icel-1] = 0.0;
             D[icel-1] = 0.0;
             indexLnew[icel]=indexLnew_org[icel];
             indexUnew[icel]=indexUnew_org[icel];

             for(j=indexLnew_org[icel-1];j<indexLnew_org[icel];j++){
                 itemLnew[j]=0;
                 ALnew[j] = 0.0;
             }
             for(j=indexUnew_org[icel-1];j<indexUnew_org[icel];j++){
                 itemUnew[j]=0;
                 AUnew[j] = 0.0;
             }
         }
     }
  }
```

Pages are allocated at the local memory of each thread

A very common technique in OpenMP program for optimization is to initialize data in parallel using the same loop schedule as will be used later in the computations.

Results: reoder0, L3-rsol0

```
128 128 128                 NX/NY/NZ
1.00e-0 1.00e-0 1.00e0      DX/DY/DZ
1.0e-08                     OMEGA, EPSICCG
36                          PEsmpTOT
-1                          NCOLORtot
0                           NFLAG
0                           METHOD
```

- 18 cores 2.94 sec
- 32 cores
  - NFLAG=0 2.51 sec
  - NFLAG=1 2.05 sec
- 36 cores
  - NFLAG=0 2.62 sec
  - NFLAG=1 1.96 sec

Programming Exercise for Report

- Implement “Sequential Reordering”
- Optimize “Two-Socket” cases with 36 cores by “First Touch Data Placement”
- Evaluate the performance of the code, and make comparison with that of the “Coalesced” code
- Original “Coalesced” Code
  - **`/lustre/t07XXYYY/multicore/L3/src0`**

- Questions in the Class on July 23

Report

- Deadline: 17:00 August 21 (Tue), 2018
- Send files via e-mail at **nakajima(at)cc.u-tokyo.ac.jp**
- Report
  - Cover Page: Name and ID must be written.
  - No more than 20 pages including figures and tables (A4).
    - Strategy
    - Structure of the Program, Details of Modification
    - Validation
    - Numerical Experiments
    - Performance Analysis
    - Remarks
  - Source list of the entire program (not included in the 20 pages above)