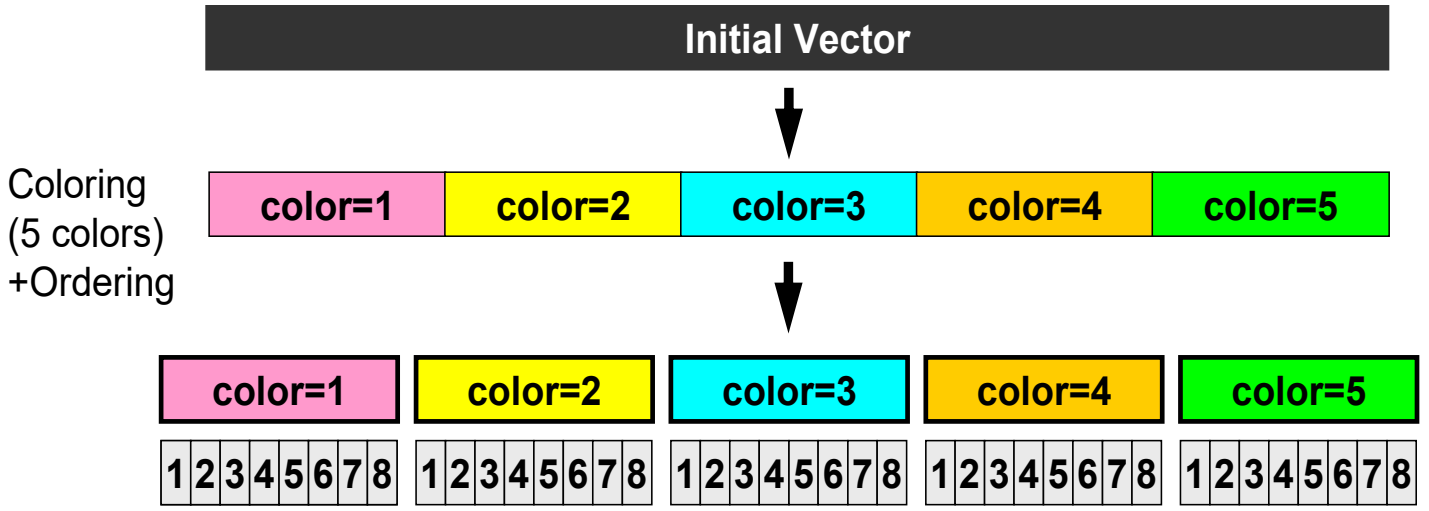
Introduction to Parallel Programming for Multicore/Manycore Clusters

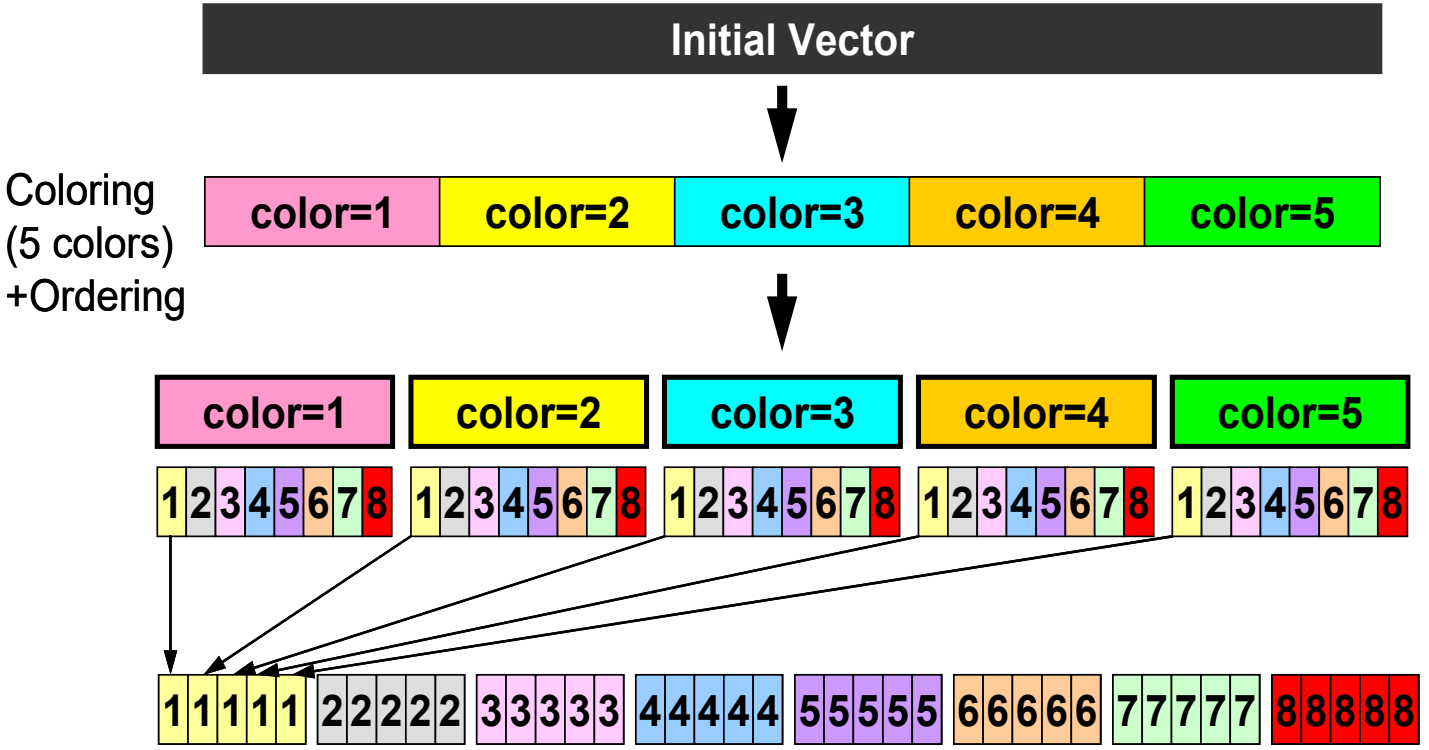
Part VI: Report

Kengo Nakajima
Information Technology Center

Coalesced



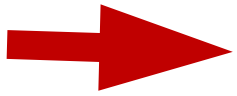
Sequential



CRS & ELL

ELL (Ellpack-Itpack): Fixed Loop Length, Effect of Prefetching

$$\begin{bmatrix} 1 & 3 & 0 & 0 & 0 \\ 1 & 2 & 5 & 0 & 0 \\ 4 & 1 & 3 & 0 & 0 \\ 0 & 3 & 7 & 4 & 0 \\ 1 & 0 & 0 & 0 & 5 \end{bmatrix}$$



1	3	
1	2	5
4	1	3
3	7	4
1	5	

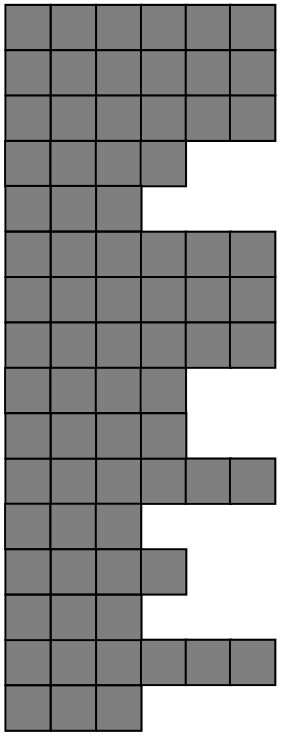
(a) CRS

1	3	0
1	2	5
4	1	3
3	7	4
1	5	0

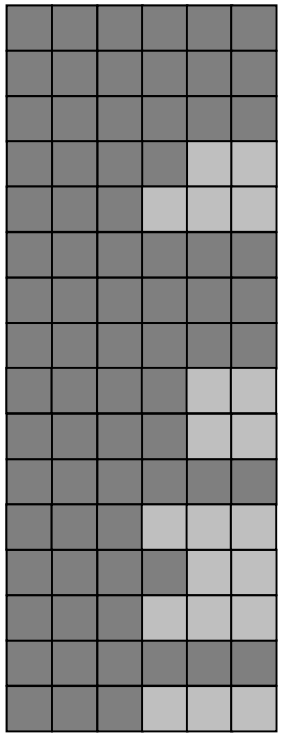
(b) ELL

Storing Formats for Sparse Matrices

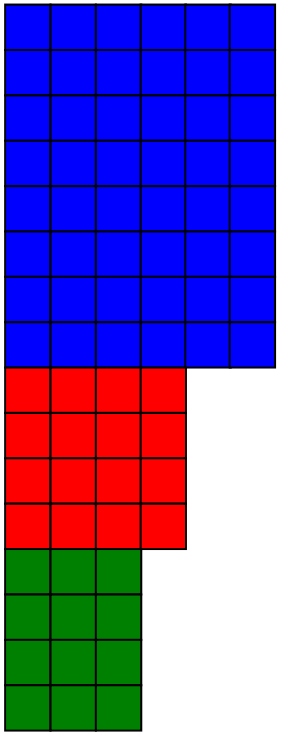
- ELL/Sliced ELL: Excellent Performance by Prefetching
- SELL-C-σ: Suitable for Vector/SIMD Processing



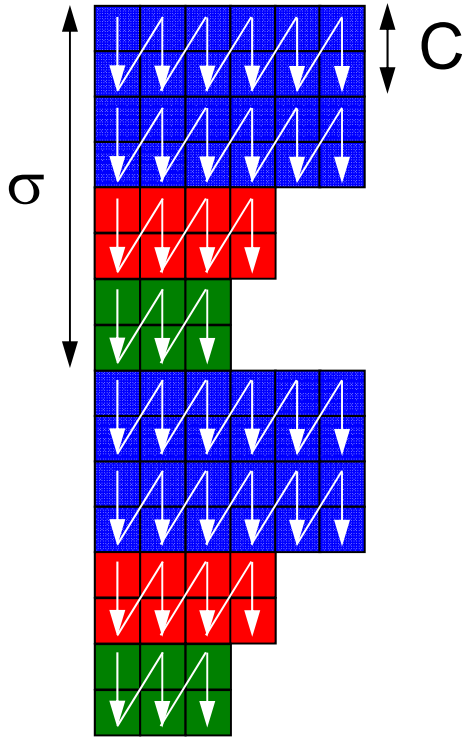
CRS



ELL



Sliced ELL



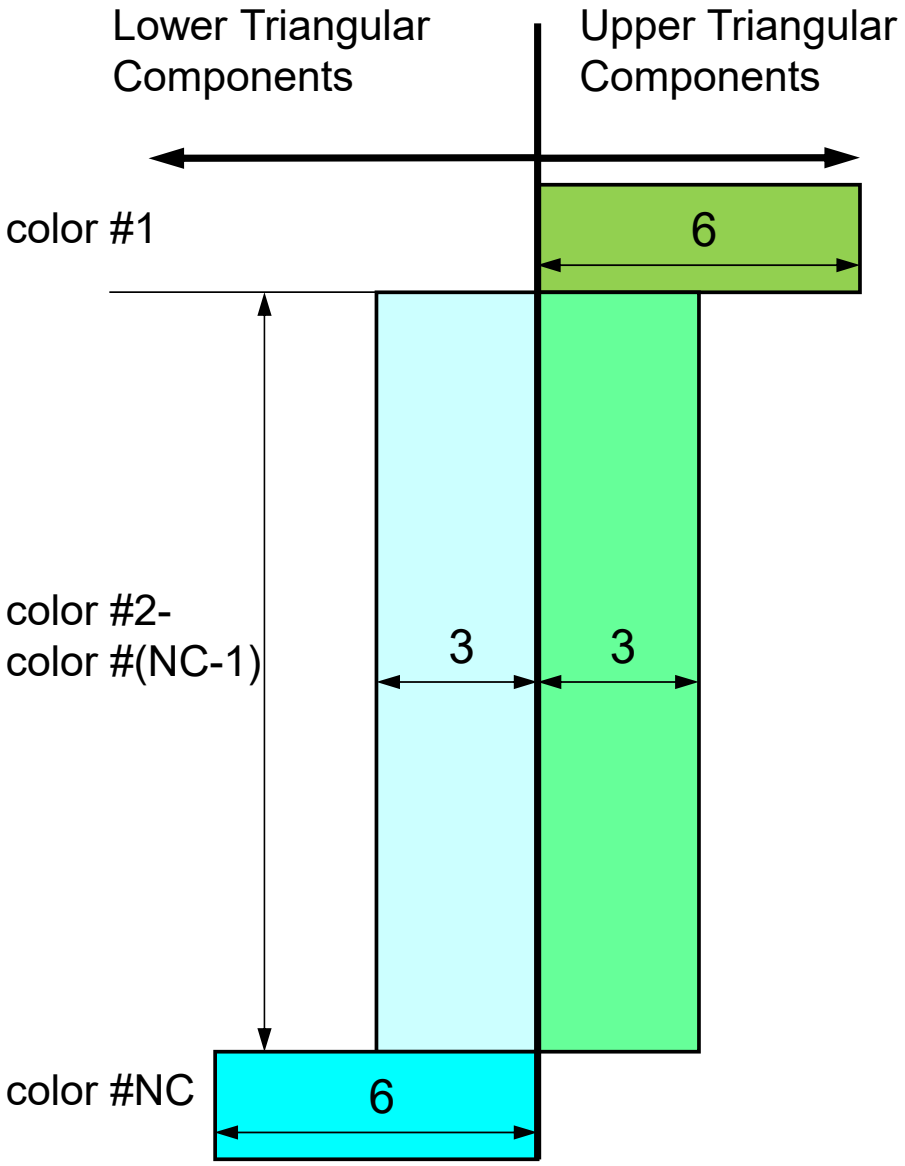
SELL-C-σ
(SELL-2-8)

ELL in Poisson3D

Not so difficult because of CM-RCM

Small difference between "original" and "sliced/modified" ELL's (just focusing on "modified" ELL)

Exception:
1st and Last Colors



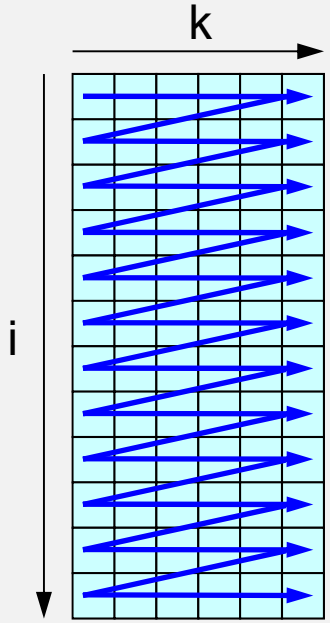
	Numbering	Matirx	Outer Loops	etc.
AR-0	Coalesced	CRS	Row-wise	
AR-1		Sliced ELL		
AC-1			Column-wise	
AC-2				Blocking
BR-0	Sequential	CRS	Row-wise	
BR-1		Sliced ELL		
BC-1			Column-wise	
BC-2				Blocking

ELL: Row-wise

CRS with fixed length

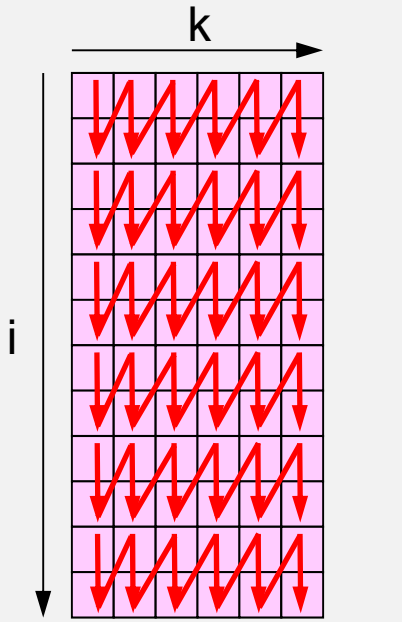
Forward Substitution

```
!$omp parallel
    do icol= 1, NCOLORtot
!$omp do
    do ip  = 1, PEsmpTOT
      do i= Index(ip-1,icol)+1, Index(ip,icol)
        do k= 1, 6
          Z(i)= Z(i) - AML(k,i)*Z(IAML(k,i))
        enddo
        Z(i)= Z(i) / DD(i)
      enddo
    enddo
    enddo
!omp end parallel
```



ELL: Column-wise
Jagged Diagonal
Forward Substitution

```
!$omp parallel
    do icol= 1, NCOLORtot
!$omp do
    do ip  = 1, PEsmpTOT
      do k= 1, 6
        do i= Index(ip-1,icol)+1, Index(ip,icol)
          Z(i)= Z(i) + AML (i,k)*Z(IAML(i,k))
        enddo
      enddo
      do i= Index(ip-1,icol)+1, Index(ip,icol)
        Z(i)= Z(i) / DD(i)
      enddo
    enddo
    enddo
!omp end parallel
```



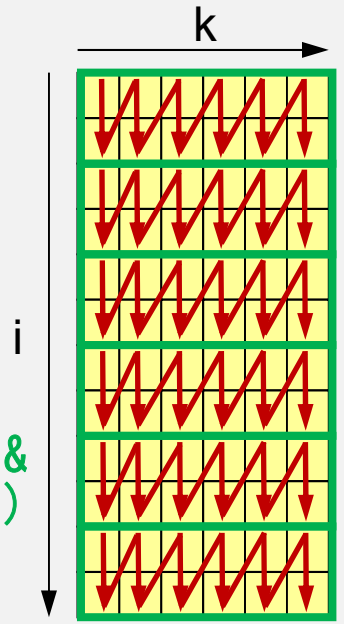
Each Color/Thread

ELL: Column-wise

Jagged Diagonal, Blocked Version

Forward Substitution

```
!$omp parallel
    do icol= 1, NCOLORtot
!$omp do
    do ip  = 1, PEsmpTOT
      blkID= (ip-1)*NCOLORtot + ip
      do k= 1, 6
      do i= IndexB(ip-1,blkID,icol)+1, &
            IndexB(ip  ,blkID,icol)
        locID= i - IndexB(ip-1,blkID,icol)
        Z(i)= Z(i) +                                          &
              AMLb(locID,k,blkID)* X(IAMLb(locID,k,blkID))
      enddo
      enddo
      do i= IndexB(ip-1,blkID,icol)+1, IndexB(ip  ,blkID,icol)
        Z(i)= Z(i) / DD(i)
      enddo
    enddo
    enddo
!omp end parallel
```



Performance Evaluation

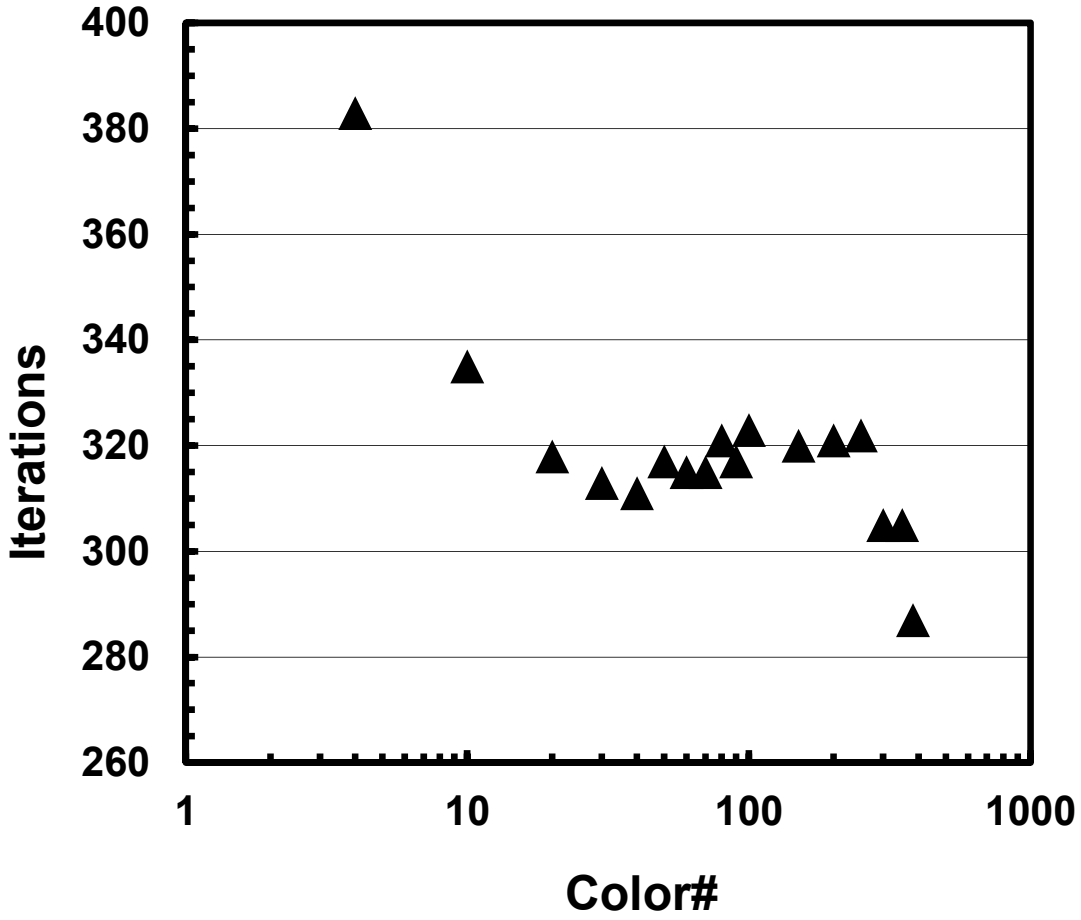
- 2,097,152 meshes (=128^3)
- Hardware
  - Intel Xeon Phi 5110P (Knights Corner)
    - (60+1) cores, 240 threads
  - Intel Xeon Phi 7210 (Knights Landing)
    - (62+2) cores, 62-124 threads used, Flat+Quadrant
    - Test node for preliminary evaluation
    - Different from the CPU's from Oakforest-PACS
  - Intel Broadwell-EP (Reedbush-U at the University of Tokyo)

Code Name	KNC	KNL-0	BDW
Architectures	Intel Xeon Phi 5110P (Knights Corner)	Intel Xeon Phi 7210 (Knights Landing)	Intel Xeon E5-2695 v4 (Broadwell-EP)
Frequency (GHz)	1.053	1.30	2.10
Max. Core # (Thread #)	60 (240)	64 (256)	18 (36)
Peak Performance (GFLOPS)	1,010.9	2,662.4	604.8
RAM (GB)	8	MCDRAM: 16 DDR4: 96	128
Peak Memory Bandwidth (GB/s)	320	-	76.8
STREAM Triad (GB/s)	159	MCDRAM: 454 DDR4: 72.5	64
Out-of-Order	N	Y	Y
	60 cores (240 threads) are used	62 cores (62/124 threads are used)	

Number of Iter's until Convergence

Better convergence for larger number of colors

Synchronization overhead is significant for larger number of colors



Comp. Time for ICCG

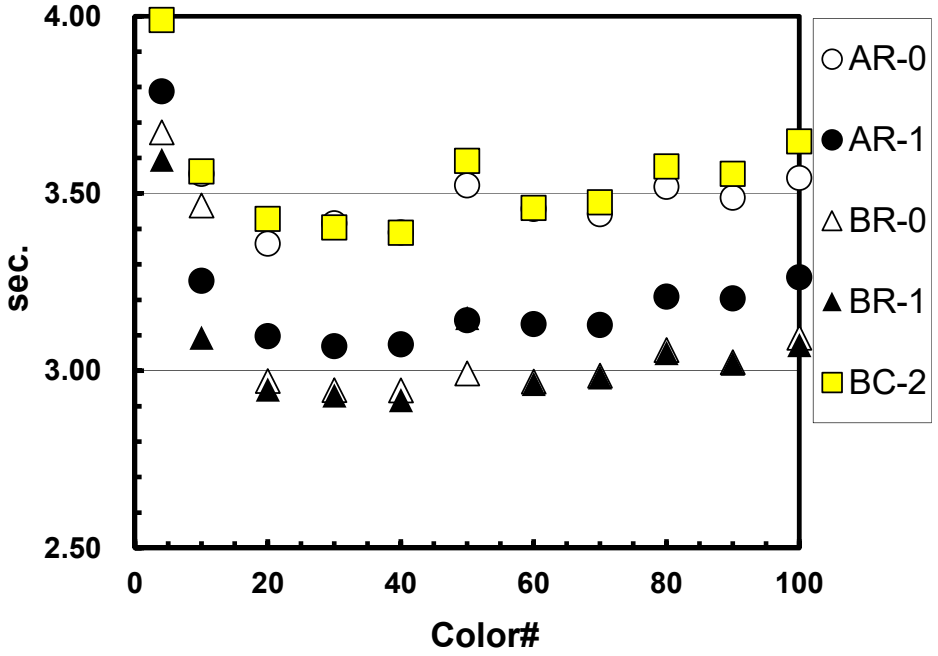
Similar Behavior

Best: BR-1 (Sequential, ELL, Row-Wise)

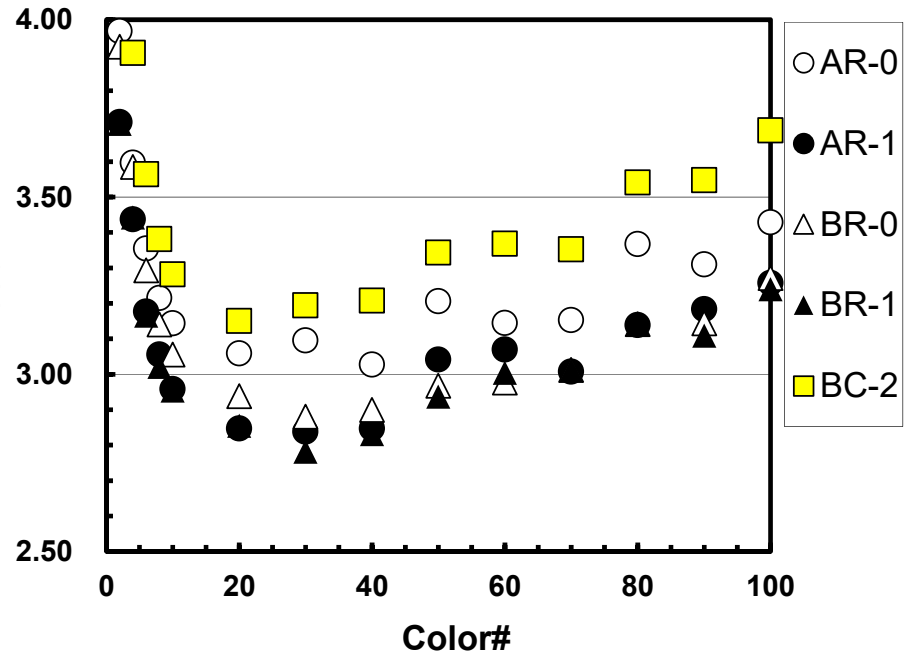
Worst: BC-2: (Sequential, ELL, Col-Wise Blocked)

Down is Good !!

BDW



KNL-0-DDR4



Comp. Time for ICCG

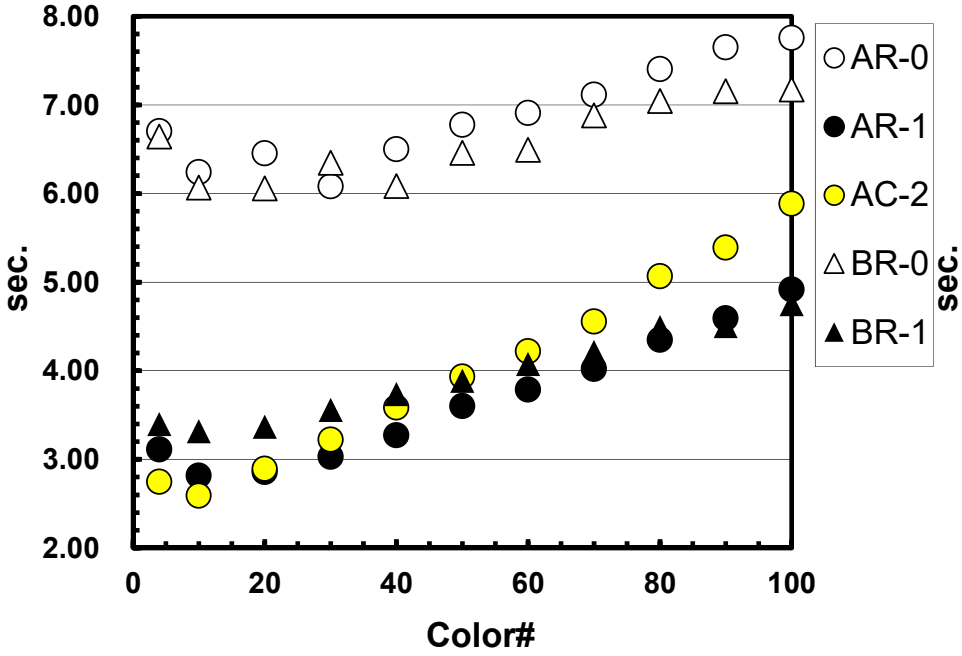
Best/KNC: AC-2 (Coalesced, ELL, Col-Wise Blocked)

Best/KNL-0: BR-1, BC-2

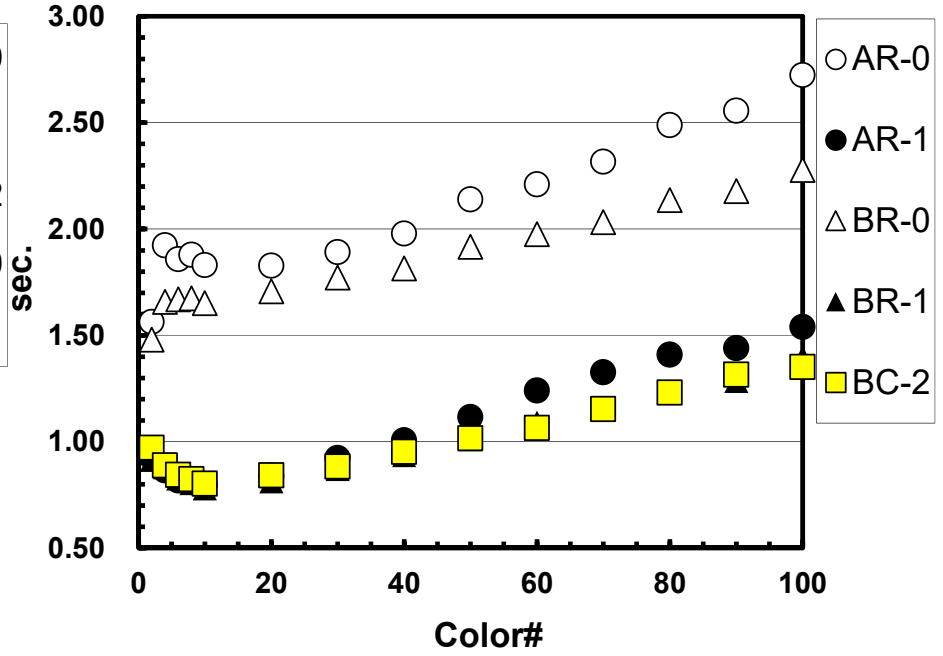
Effect of ELL is Significant

Down is Good !!

KNC



KNL-0-MCDRAM

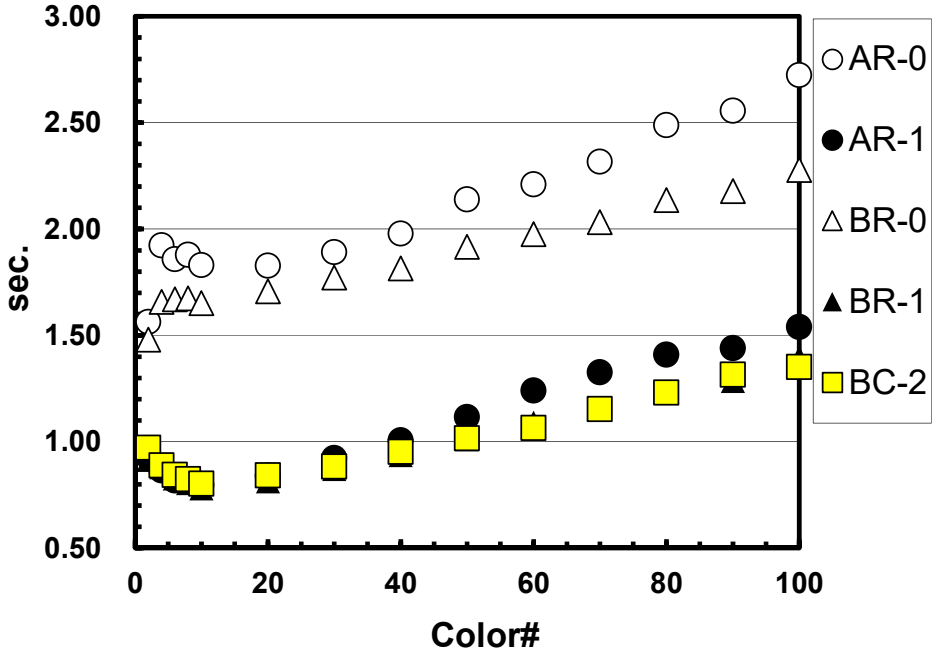


Comp. Time for ICCG

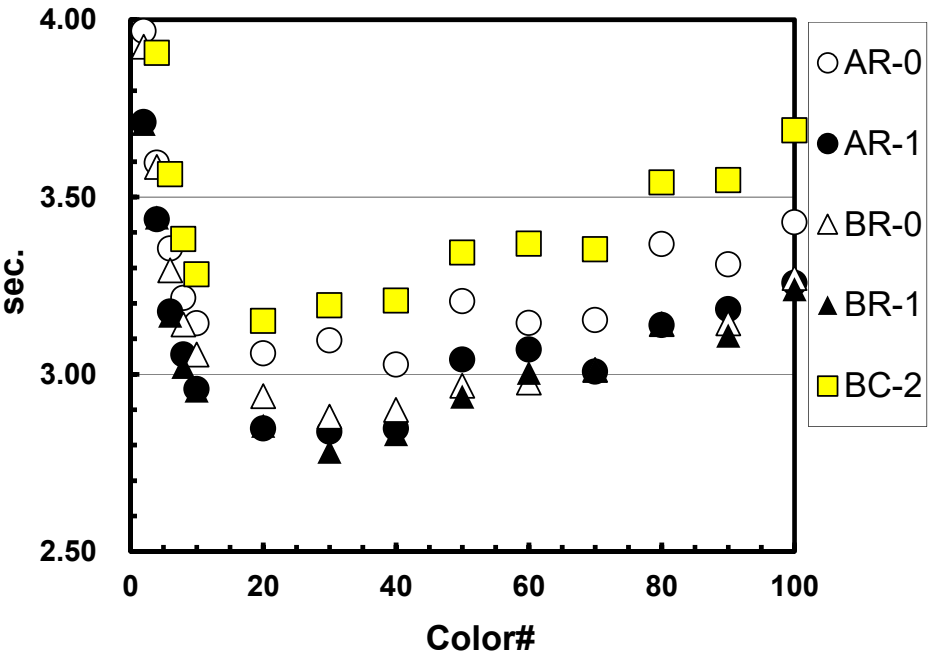
Effect of ELL is NOT so significant for DDR4
BC-2 (suitable for vectorization) is the best for MCDRAM, but the worst for DDR4

Down is Good !!

KNL-0-MCDRAM



KNL-0-DDR4

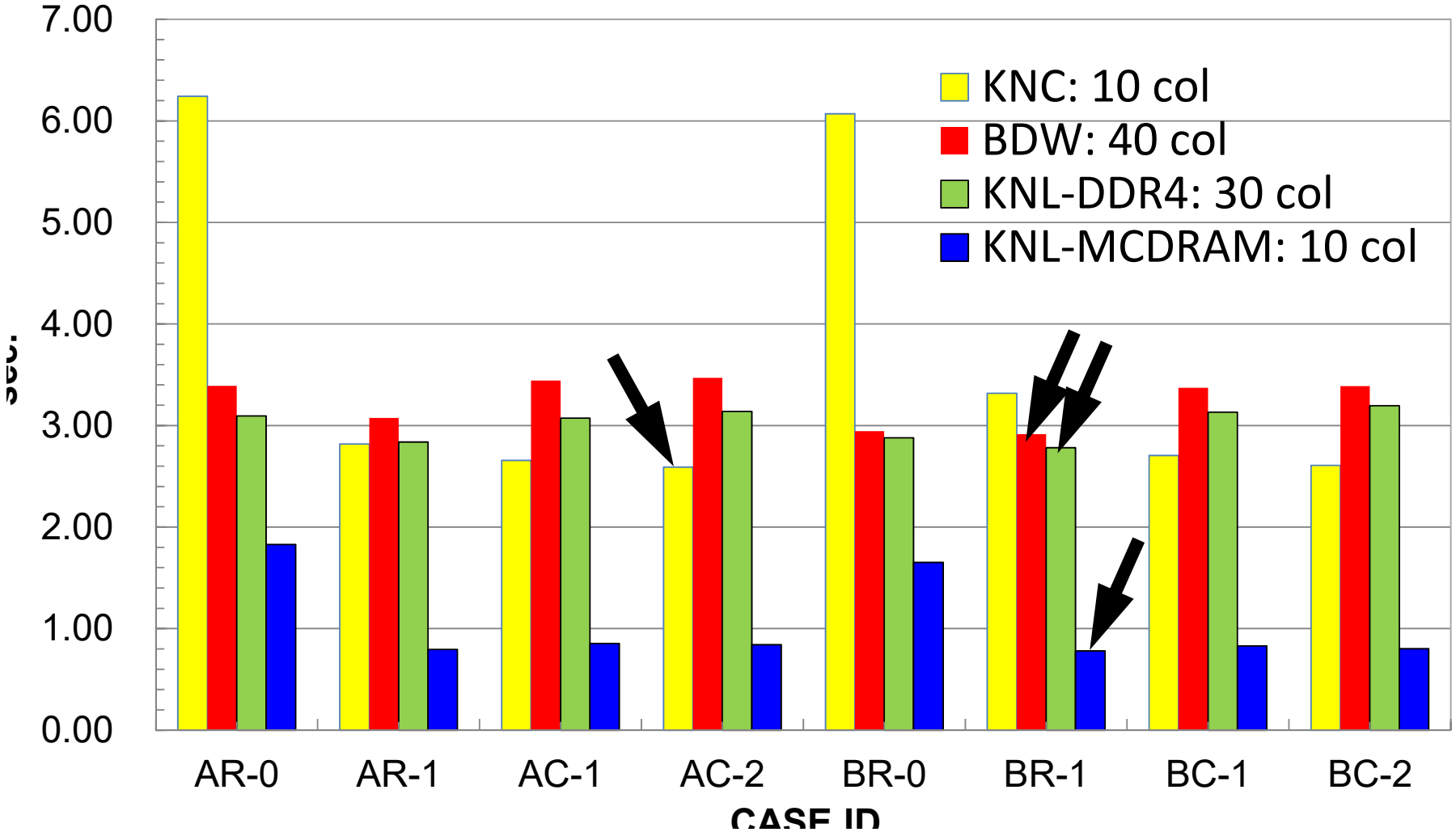


Comp. Time for ICCG (Opt. Color #)

Best Case: KNC: AC-2, BC-2, BDW: BR-1

KNL-0-DDR4/MCDRAM: BR-1

Down is Good !!

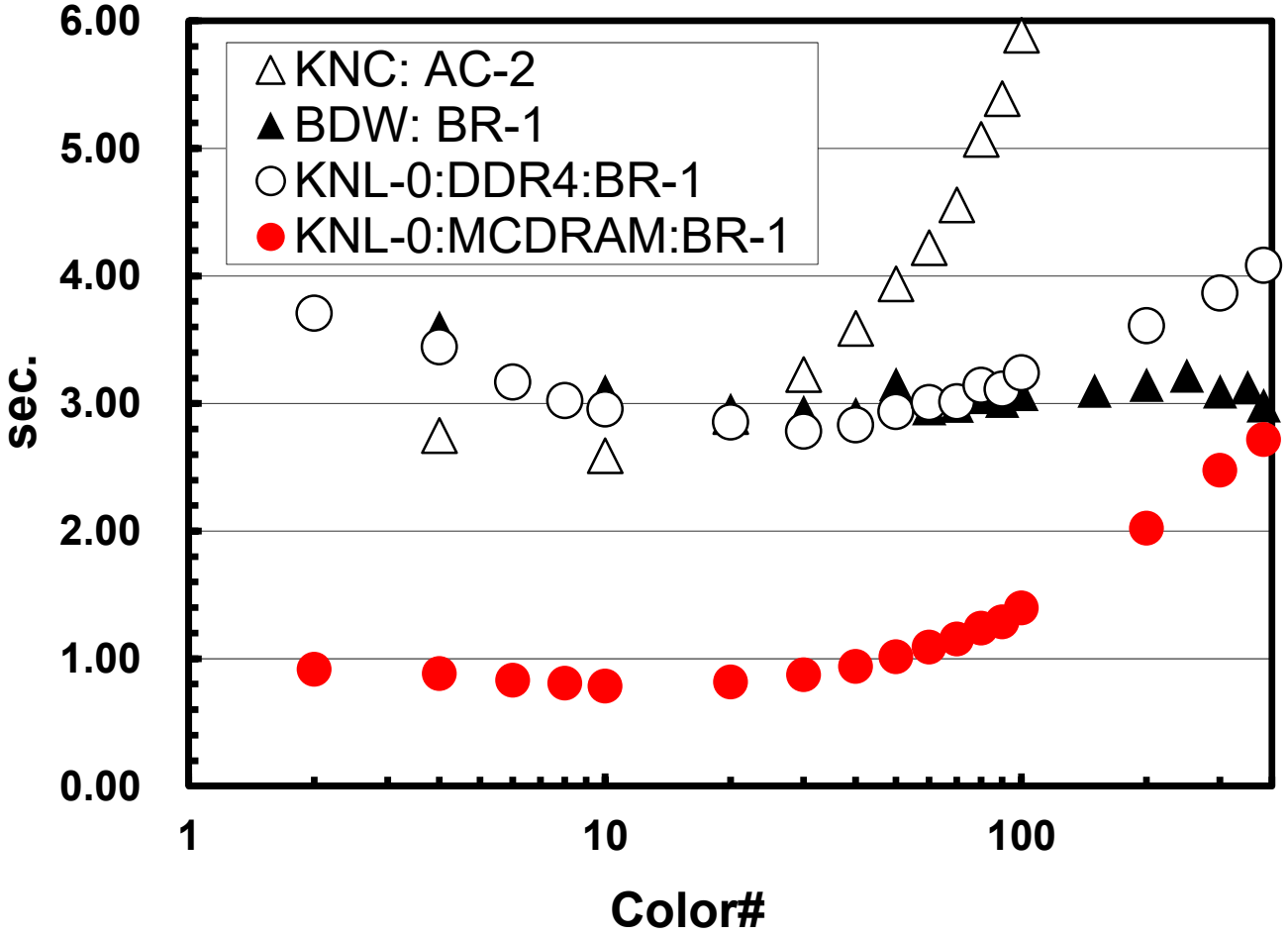


Comp. Time for ICCG (Best ELL)

Effects of synchronization overhead are significant on KNL & KNC, if number of colors is larger

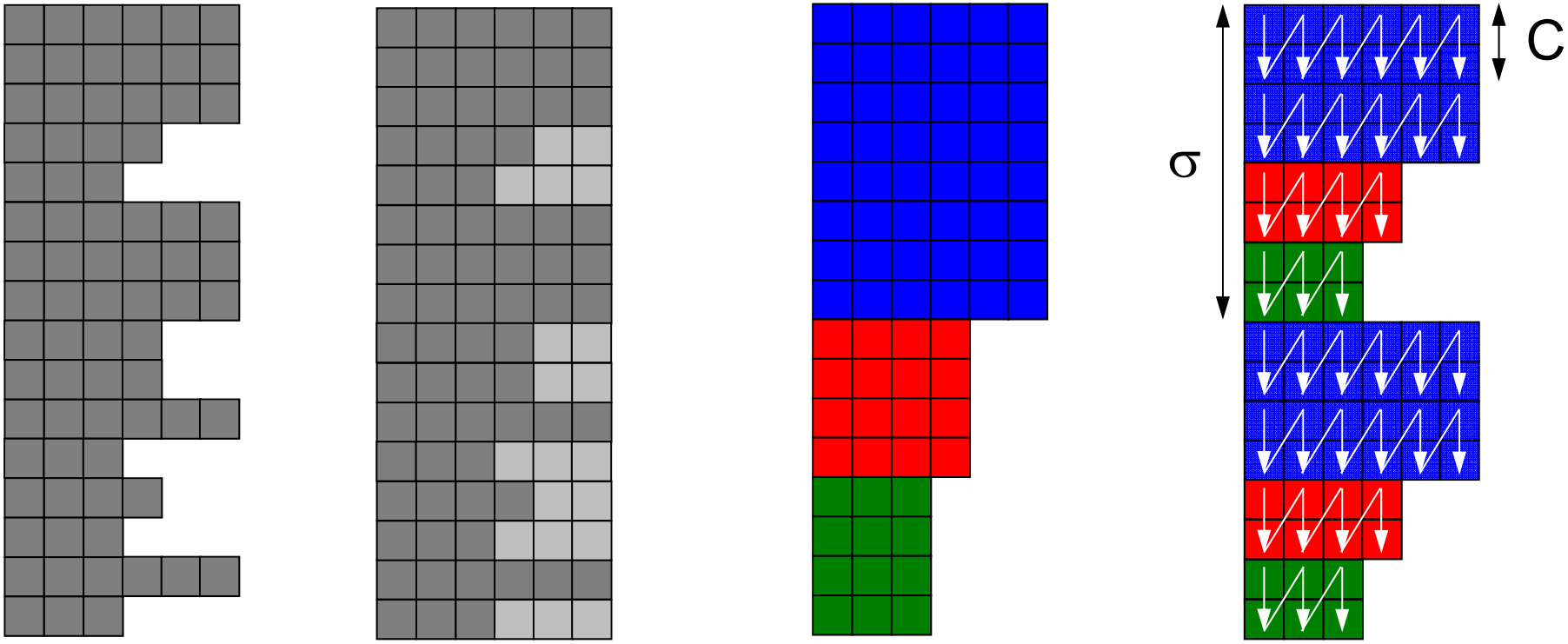
Generally, optimum number of color is 10 for KNL/KNC

Down is Good !!



Storing Formats for Sparse Matrices

- ELL/Sliced ELL: Excellent Performance by Prefetching
- SELL-C-σ: Good for Vector/SIMD (by FAU/Erlangen)
  - This work is the first example where SELL-C-σ is applied to IC/ILU preconditioning (FAU is also trying).



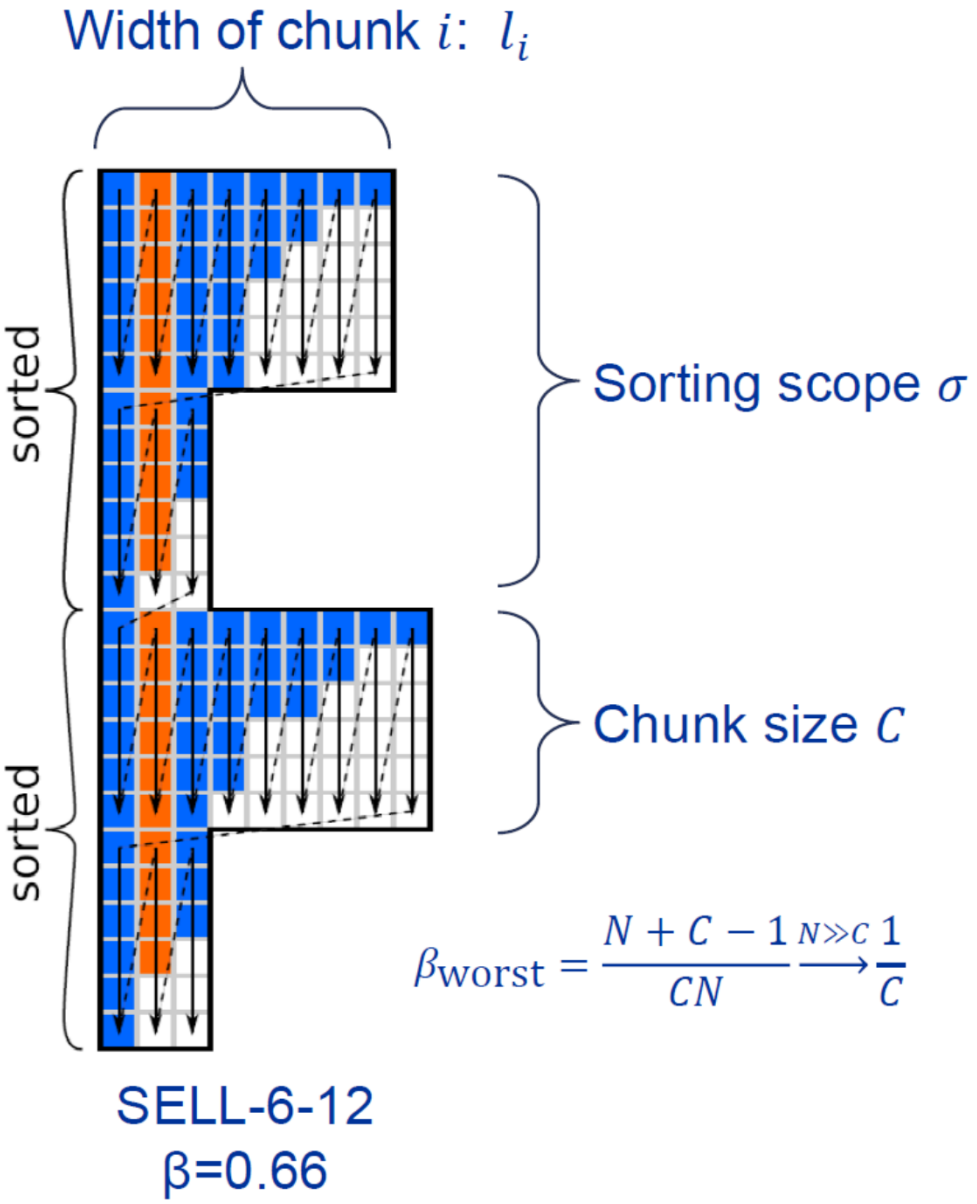
CRS ELL Sliced ELL SELL-C-σ (SELL-2-8)

Constructing SELL-C-σ

1. Pick chunk size C (guided by SIMD/T widths)
2. Pick sorting scope σ
3. Sort rows by length within each sorting scope
4. Pad chunks with zeros to make them rectangular
5. Store matrix data in “chunk column major order”

“Chunk occupancy”: fraction of “useful” matrix entries

$$\beta = \frac{N_{nz}}{\sum_{i=0}^{N_c} C \cdot l_i}$$



[Kreutzer, Hager, Wellein 2014]

SELL-8-1 for Forward Subst.

```
!$omp parallel private(ic,ip,ip0,iq0,iq1,iq2,ib,ib0,is,i,k)
...
      do ic= 2, NCOLORtot-1
!$omp do
        do ip= 1, PEsmpTOT
          iq1= SMPindex((ic-1)*PEsmpTOT + ip-1) + 1
          iq2= SMPindex((ic-1)*PEsmpTOT + ip)
          ip0= (ic-1)*PEsmpTOT + ip
          iq0= (iq1-1)*8
          do ib  = iq1, iq2
            ib0= (ib-iq1)*8
          do k= 1, 3
!$omp simd
          do is  = 1, 8
            i= iq0 + ib0 + is
            W(i,Z)= W(i,Z)- AL(ib0+is,k,ip0)*W(itemL(ib0+is,k,ip0),Z)
          enddo
          enddo
!$omp simd
          do is  = 1, 8
            i= iq0 + ib0 + is
            W(i,Z)= W(i,Z) * W(i,DD)
          enddo
          enddo
        enddo
      enddo
...
!$omp end parallel
```

	Numbering	Matirx	Outer Loops	etc.
AR-0	Coalesced	CRS	Row-wise	
AR-1		Sliced ELL		
AC-1			Column-wise	
AC-2				Blocking
AC-3		SELL-C-σ		
BR-0	Sequential	CRS	Row-wise	
BR-1		Sliced ELL		
BC-1			Column-wise	
BC-2				Blocking
BC-3		SELL-C-σ		

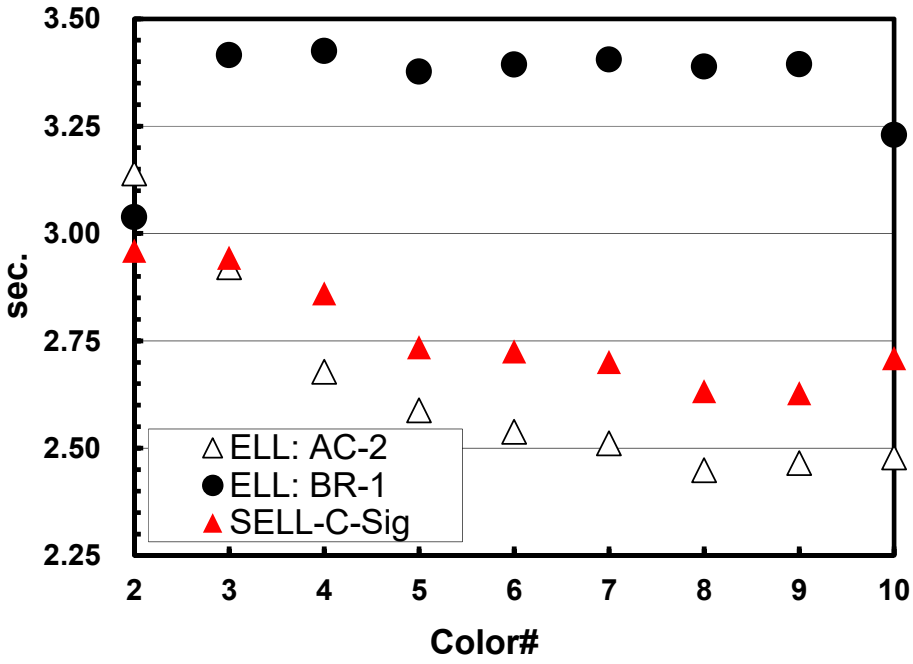
Computation Time for ICCG Solver

2-10 Colors, KNL-0 is 3x faster than KNC

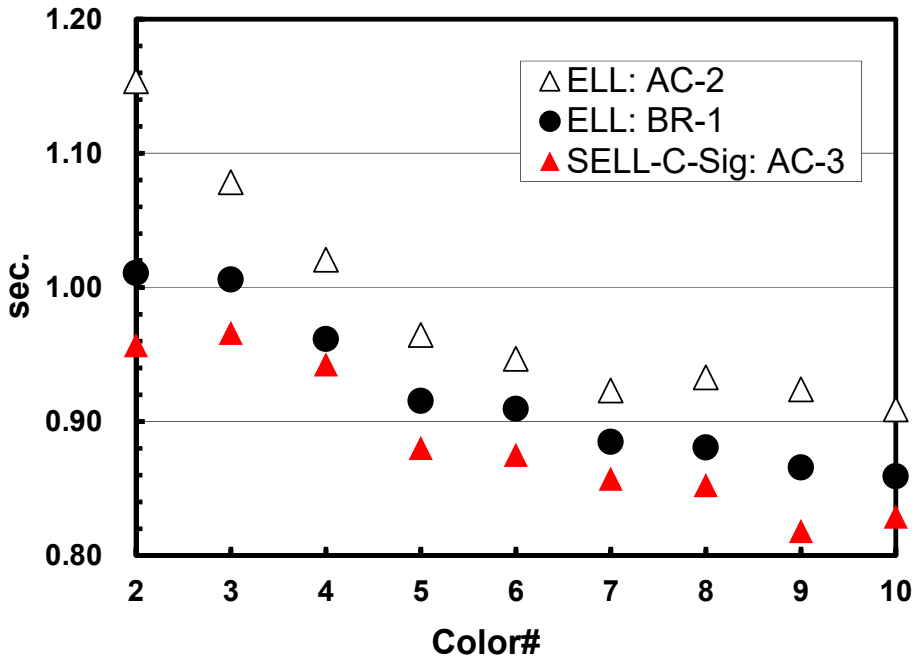
△:AC-2, ●:BR-1, ▲:SELL-C-σ (SELL-8-1)

Down is Good !!

KNC



KNL-0-MCDRAM

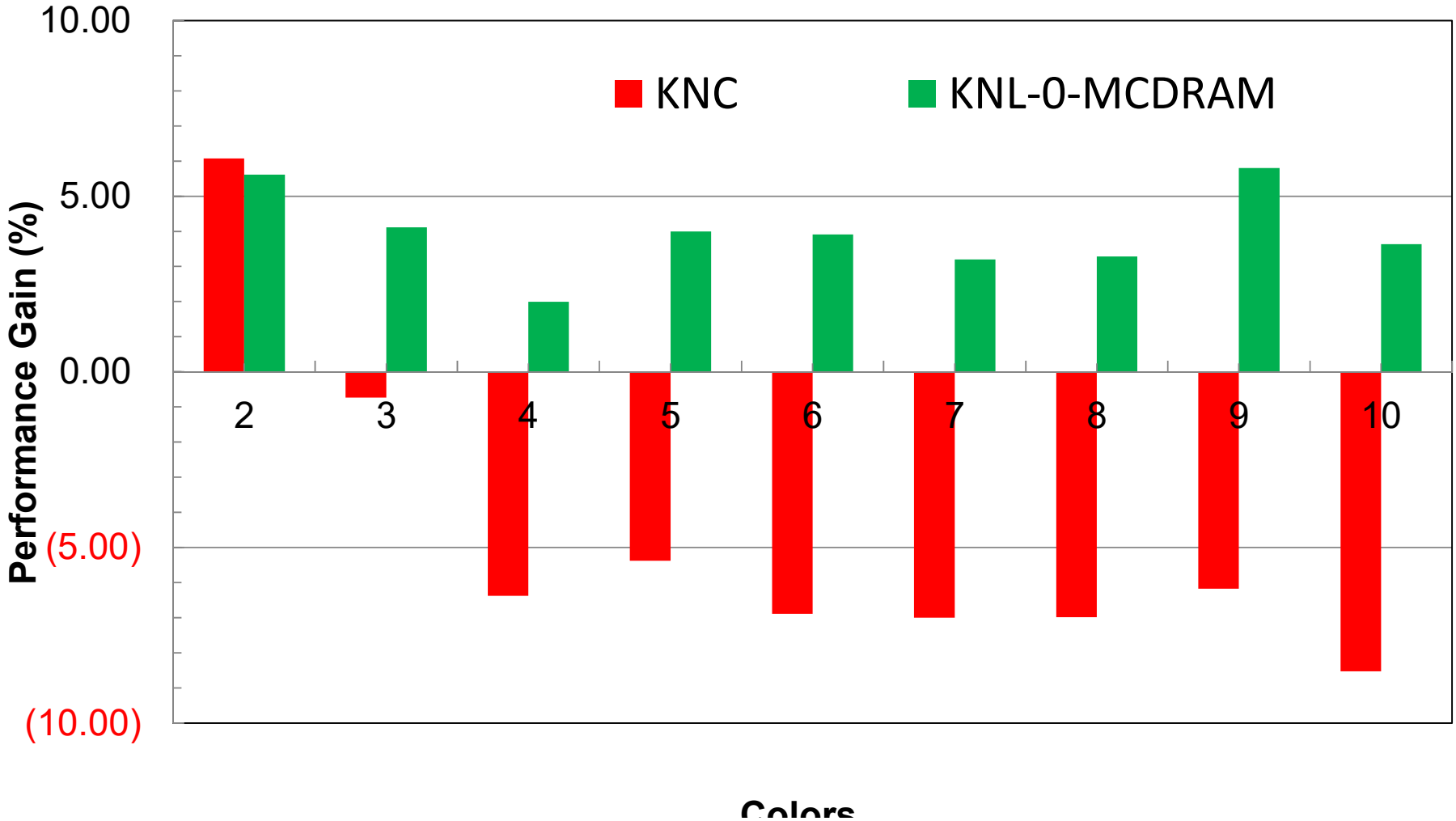


Effects of SELL-C-σ on KNL-0-MC/D

Improvement over Original Best ELL Cases (2-10 colors)

SELL-C-σ is rather slower on KNC

Fig.15 is wrong



Summary

- ICCG Solver on KNC/KNL
  - KNL: Flat, Quadrant, MCDRAM/DDR4 only
- CM-RCM Reordering
  - Optimum number of color is 10 on KNC/KNL
- Effects of ELL/SELL-C-σ is significant in KNC/KNL
  - 2x faster than CRS
- KNL with MCDRAM is 3x faster than KNC
  - Peak Performance
  - Memory Bandwidth of MCDRAM
    - KNL and KNC are competitive for DDR4 only case
- Improvement of Performance (e.g. 1.5% of peak for HPCG at 8,192 nodes of OFP)
  - Blocking, Further Vectorization

Effects of SELL-C-σ

- The first case applied to IC/ILU preconditioning

- KNC: slower than ELL: overhead, extra computations
- KNL: slightly faster than ELL

- Number of operations for non-zero off-diagonals at each row is very small
  - 3 for lower/upper components
    - Exception: This number is 6, if the color number is 2
    - This is the only case where SELL-C-σ is faster than ELL on KNC
  - FEM cases (e.g. 27-point stencil) should be evaluated

Programming Exercise for Report

- Target: **`multicore/omp/src20`**
  - PCG with Diagonal Scaling Preconditioning
  - Apply “ELL” matrix storage format to the target

- Comparison of performance
- Row-Wise
- Column-Wise

- Examples of “Row-Wise” approaches are shown in the following pages.

Example: poi_gen

For all rows, number of non-zero off-diagonal components= 6（=3+3）

```
!C
!C-- 1D array

      allocate (indexL(0:nn), indexU(0:nn))
      indexL= 0
      indexU= 0

      do icel= 1, ICELTOT
        indexL(icel)= 3
        indexU(icel)= 3
      enddo

      do icel= 1, ICELTOT
        indexL(icel)= indexL(icel) + indexL(icel-1)
        indexU(icel)= indexU(icel) + indexU(icel-1)
      enddo

      NPL= indexL(ICELTOT)
      NPU= indexU(ICELTOT)

      allocate (itemL(NPL), AL(NPL))
      allocate (itemU(NPU), AU(NPU))

      itemL= 0
      itemU= 0
         AL= 0.d0
         AU= 0.d0
```

Example: poi_gen

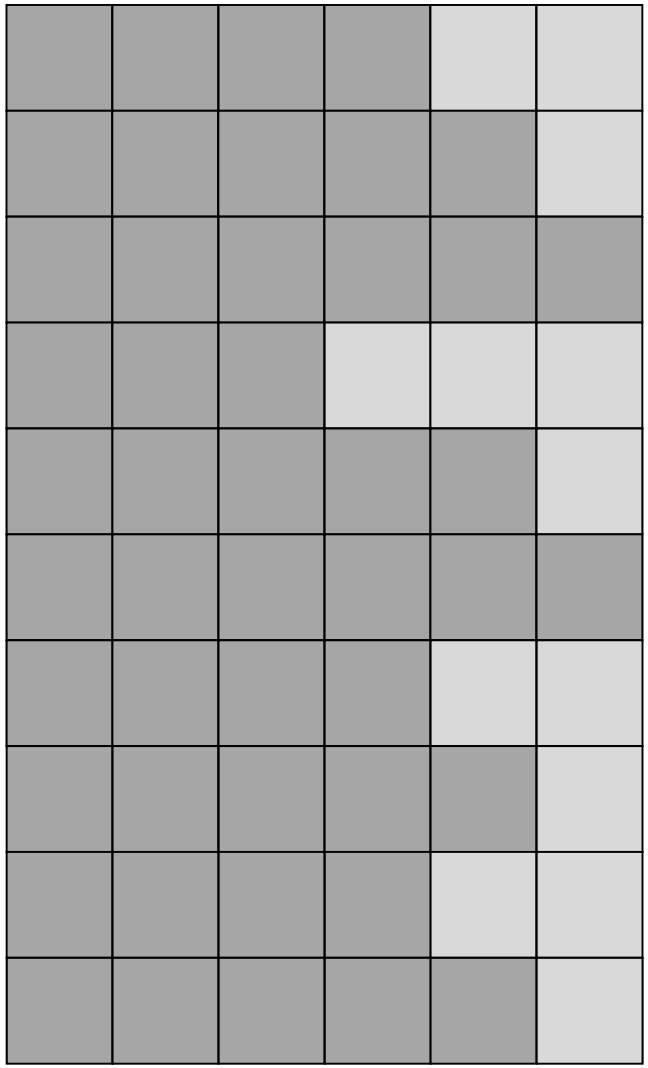
Column ID if AL/AU= 0.0: ICELTOT+icou

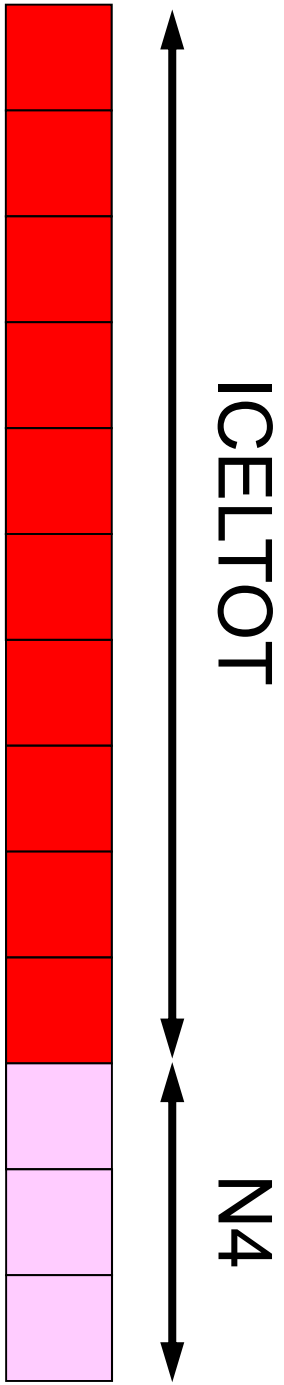
```
      icou= 0
      do i= 1, ICELTOT
      do k= 1, 3
        if (itemL(indexL(i-1)+k).eq.0) then
          icou= icou + 1
          itemL(indexL(i-1)+k)= ICELTOT + icou
        endif
        if (icou.eq.N4) icou= 0
      enddo
      enddo

      icou= 0
      do i= 1, ICELTOT
      do k= 1, 3
        if (itemU(indexU(i-1)+k).eq.0) then
          icou= icou + 1
          itemU(indexU(i-1)+k)= ICELTOT + icou
        endif
        if (icou.eq.N4) icou= 0
      enddo
      enddo
```

```
icou: 1-N4
N4: Not so big value (e.g.256)

PHI(ICELTOT+N4)
```




ICELTOT
N4

Example: solver_PCG

Implementation of Mat-Vec (1/5)

```
      do i= 1, N
        VAL= D(i)*W(i,P)

        do k= indexL(i-1)+1, indexL(i-1)+3
          VAL= VAL + AL(k)*W(itemL(k),P)
        enddo

        do k= indexU(i-1)+1, indexU(i-1)+3
          VAL= VAL + AU(k)*W(itemU(k),P)
        enddo

        W(i,Q)= VAL
      enddo
```

```
W(ICELTOT+N4,4)
```

Example: solver_PCG

Implementation of Mat-Vec (2/5)

```
      do i= 1, N
        VAL= D(i)*W(i,P)

        do k= 1, 3
          kk= (i-1)*3 + k
          VAL= VAL + AL(kk)*W(itemL(kk),P)
        enddo

        do k= 1, 3
          kk= (i-1)*3 + k
          VAL= VAL + AU(kk)*W(itemU(kk),P)
        enddo

        W(i,Q)= VAL
      enddo
```

```
W(ICELTOT+N4,4)
```

Example: solver_PCG

Implementation of Mat-Vec (3/5)

```
      do i= 1, N
        VAL= D(i)*W(i,P)

        do k= 1, 3
          kk= (i-1)*3 + k
          VAL= VAL + AL(kk)*W(itemL(kk),P)
     &             + AU(kk)*W(itemU(kk),P)
        enddo

        W(i,Q)= VAL
      enddo
```

```
      do i= 1, N
        VAL= D(i)*W(i,P)

        do k= 1, 6
          kk= (i-1)*6 + k
          VAL= VAL + AMAT(kk)*W(itemLU(kk),P)
        enddo

        W(i,Q)= VAL
      enddo
```

Example: solver_PCG: Row-Wise

Implementation of Mat-Vec (4/5)

```
do i= 1, N
  VAL= 0.d0
  do k= 1, 7
    VAL= VAL + AMAT(k,i)*W(IALU(k,i),P)
  enddo
  W(i,Q)= VAL
enddo
```

```
for (i=0; i<N, i++) {
   VAL= 0.0;
   for (k=0; k<7, k++) {
     VAL= VAL + AMAT[i][k]*W(IALU[i][k],P);
   }
   W[Q][i]= VAL;
}
```

Example: solver_PCG: Column-Wise

Implementation of Mat-Vec (5/5)

```
do i= 1, N
    W(i,Q)= 0.0
enddo
do k= 1, 7
  do i= 1, N
    W(i,Q)= W(i,Q)+ AMAT(i,k)*W(IALU(i,k),P)
  enddo
enddo
```

```
for (i=0; i<N, i++) {
   W[Q][i]= 0.0
}
for (k=0; k<7, k++) {
   for (i=0; i<N, i++) {
     W[Q][i]= W[Q][i] + AMAT[k][i]*W[IALU[k][i]][P];
   }
}
```

Report

- Deadline: 17:00 August 22 (Tue), 2017
- 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)