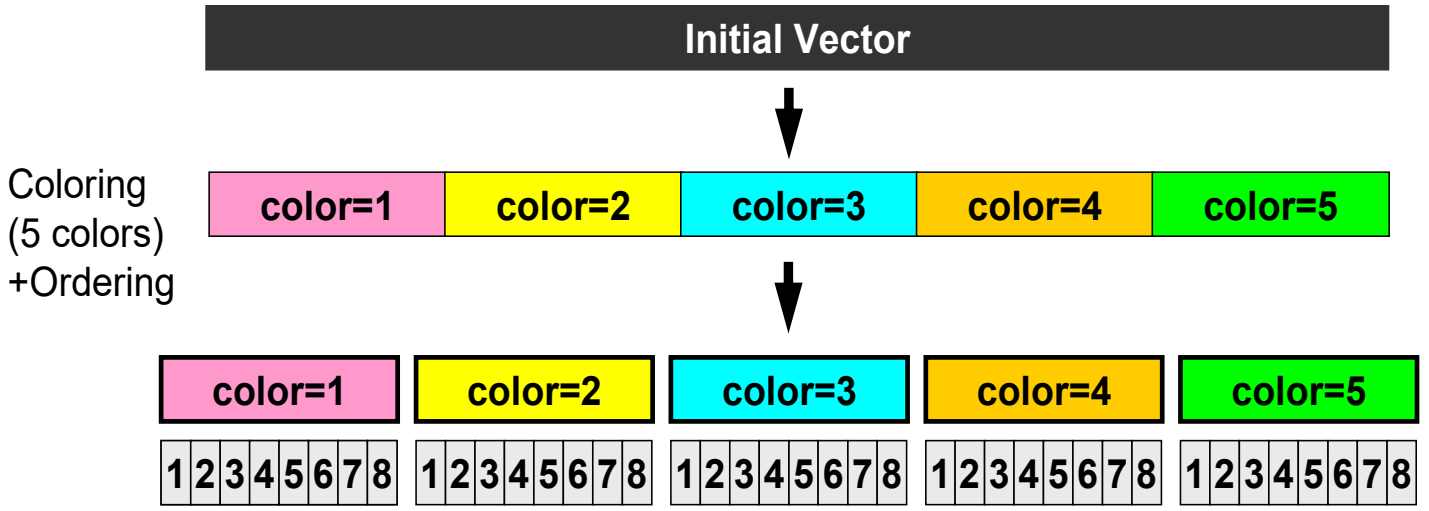
Introduction to Parallel Programming for Multicore/Manycore Clusters

Part VI: Report

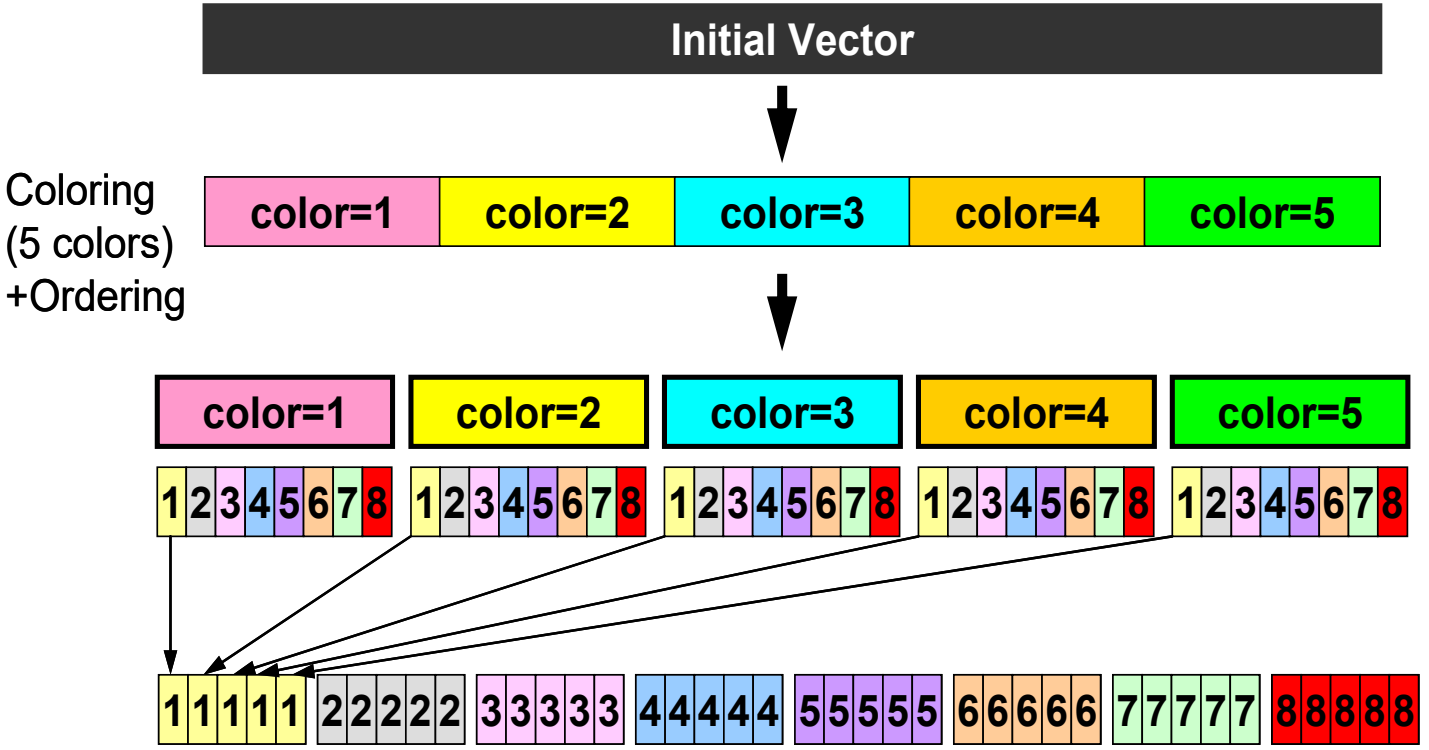
Kengo Nakajima
Information Technology Center

Code Name	FX10	MIC	IvyB
Architectures	**Fujitsu SPARC64 IX fx**	**Intel Xeon Phi 5110P (Knights Corner)**	**Intel Xeon E5-2680 v2 (Ivy-Bridge-EP)**
Frequency (GHz)	1.848	1.053	2.80
Core # (Thread #)	16 (16)	60 (240)	10 (20)
Memory Type	DDR3	GDDR5	DDR3
Peak Performance (GFLOPS)	236.5	1,010.9	224.0
RAM (GB)	32	8	64
Peak Memory Bandwidth (GB/s)	85.1	320	59.7
Cache	L1:32KB/core L2:12MB/socket	L1:32KB/core L2:512KB/core	L1:32KB/core L2:256KB/core L3:25MB/socket

Coalesced



Sequential



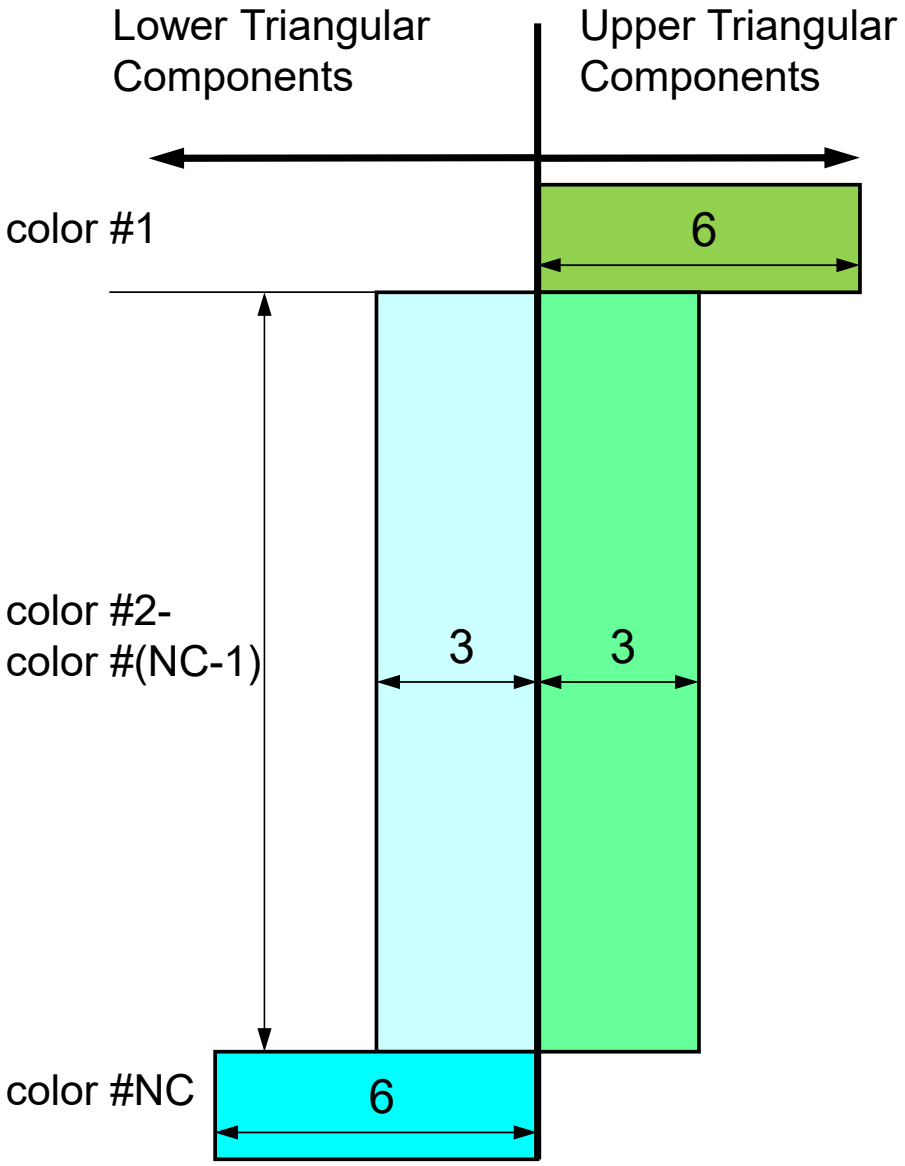
<table>
<tr><th></th><th>Numbering</th><th>Matirx</th><th>Outer Loops</th><th>etc.</th></tr>
<tr><td>AR-0</td><td rowspan="5">Coalesced</td><td>CRS</td><td rowspan="3">Row-wise</td><td></td></tr>
<tr><td>AR-1</td><td rowspan="4">Sliced ELL</td><td></td></tr>
<tr><td>AR-2</td><td>Indirect Access</td></tr>
<tr><td>AC-1</td><td rowspan="2">Column-wise</td><td></td></tr>
<tr><td>AC-2</td><td>Blocking</td></tr>
<tr><td>BR-0</td><td rowspan="4">Sequential</td><td>CRS</td><td rowspan="2">Row-wise</td><td></td></tr>
<tr><td>BR-1</td><td rowspan="3">Sliced ELL</td><td></td></tr>
<tr><td>BC-1</td><td rowspan="2">Column-wise</td><td></td></tr>
<tr><td>BC-2</td><td>Blocking</td></tr>
</table>

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

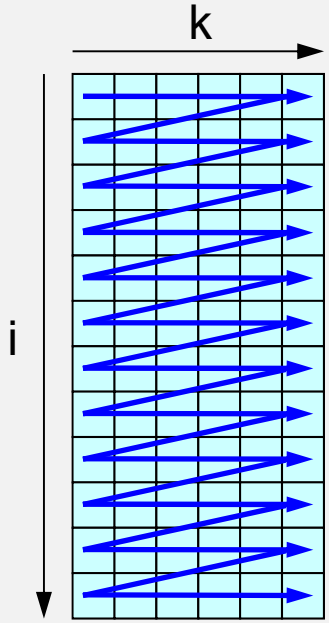


ELL: Row-wise

CRS with fixed length

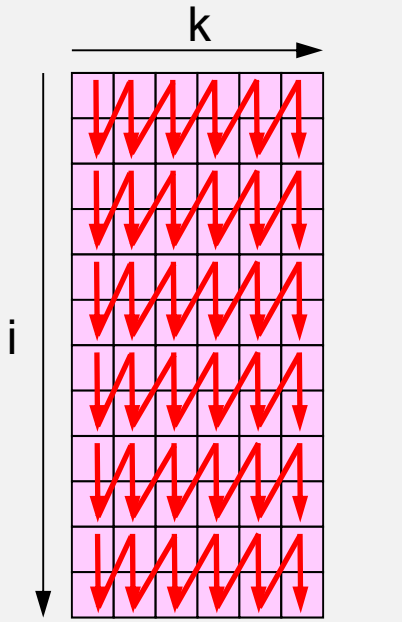
Forward Substitution

```
!$omp parallel
    do icol= 1, NCOLORtot
!$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
```



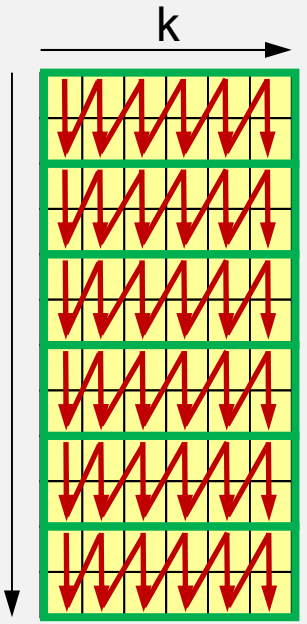
Each Color/Thread

ELL: Column-wise

Jagged Diagonal, Blocked Version

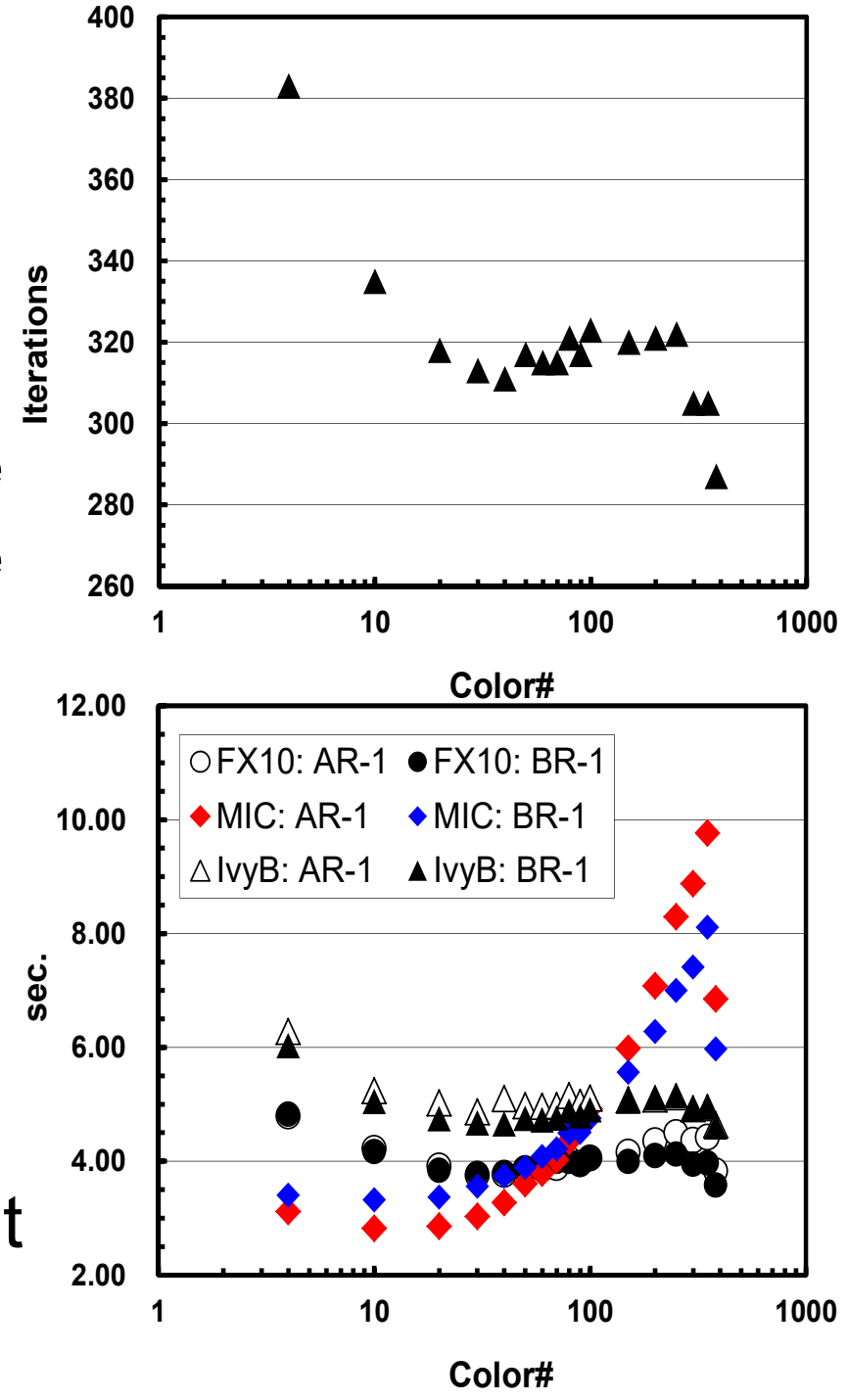
Forward Substitution

```
!$omp parallel
    do icol= 1, NCOLORtot
!$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
```



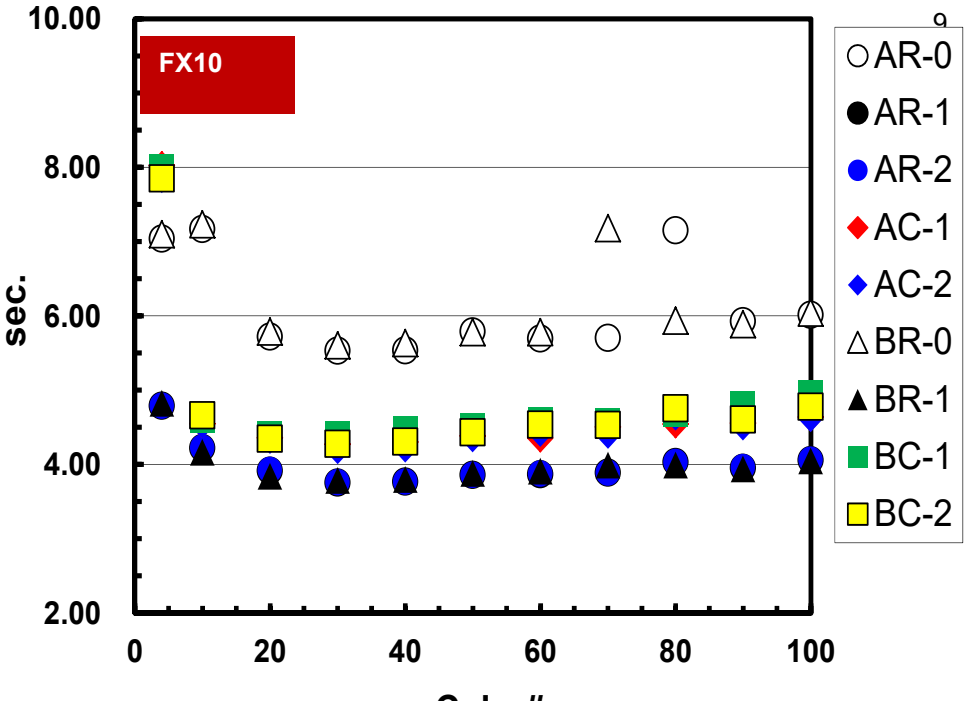
Number of Colors and Comp. Time

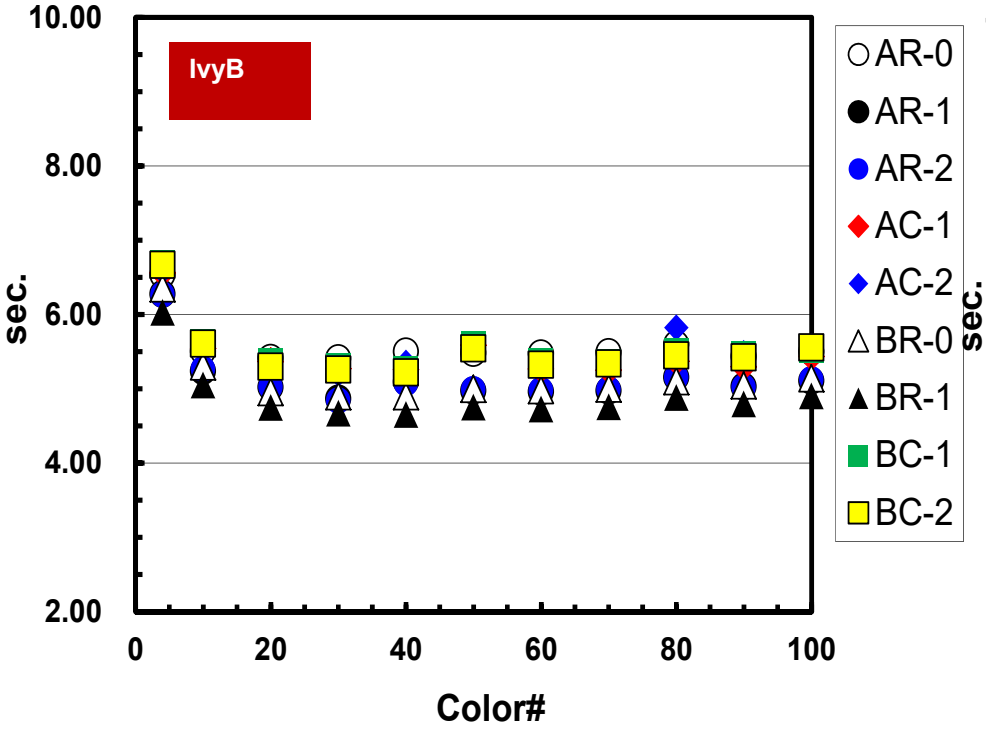
- ELL
  - AR-1: Coalesced + Row-wise
  - BR-1: Sequential + Row-wise
- More colors
  - less iterations
  - more sync. overhead
  - significant in MIC
- Optimum number of color depends on architectures
- FX10, IvyB: RCM is the best

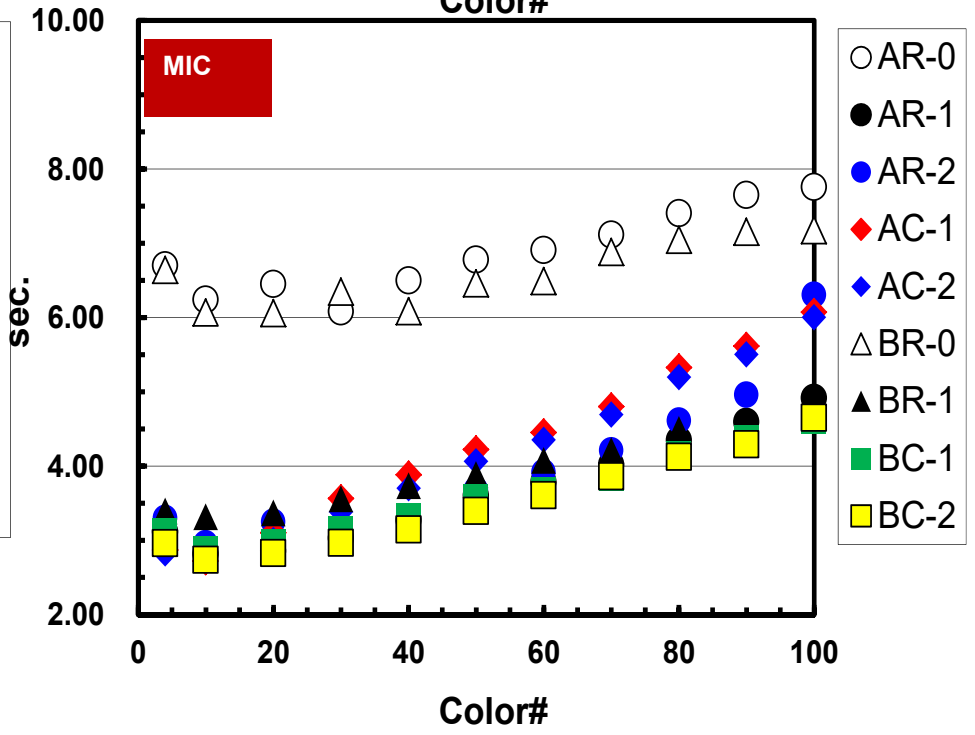


Effect of Number of Colors (~100)

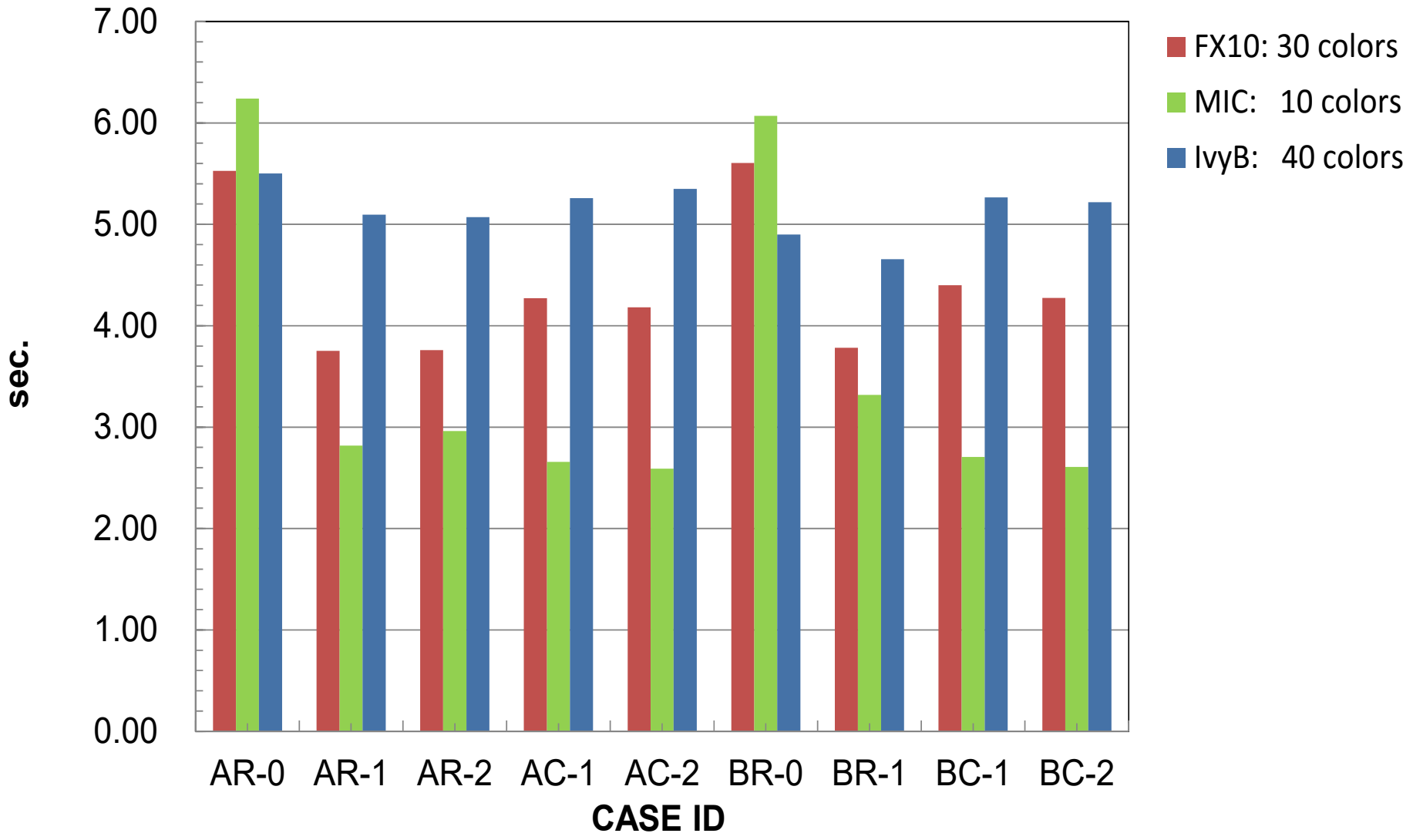
- FX10, MIC: effect of ELL is significant
- IvyB: Effect of CRS/ELL, number of colors: small
  - Out-of-Order ?



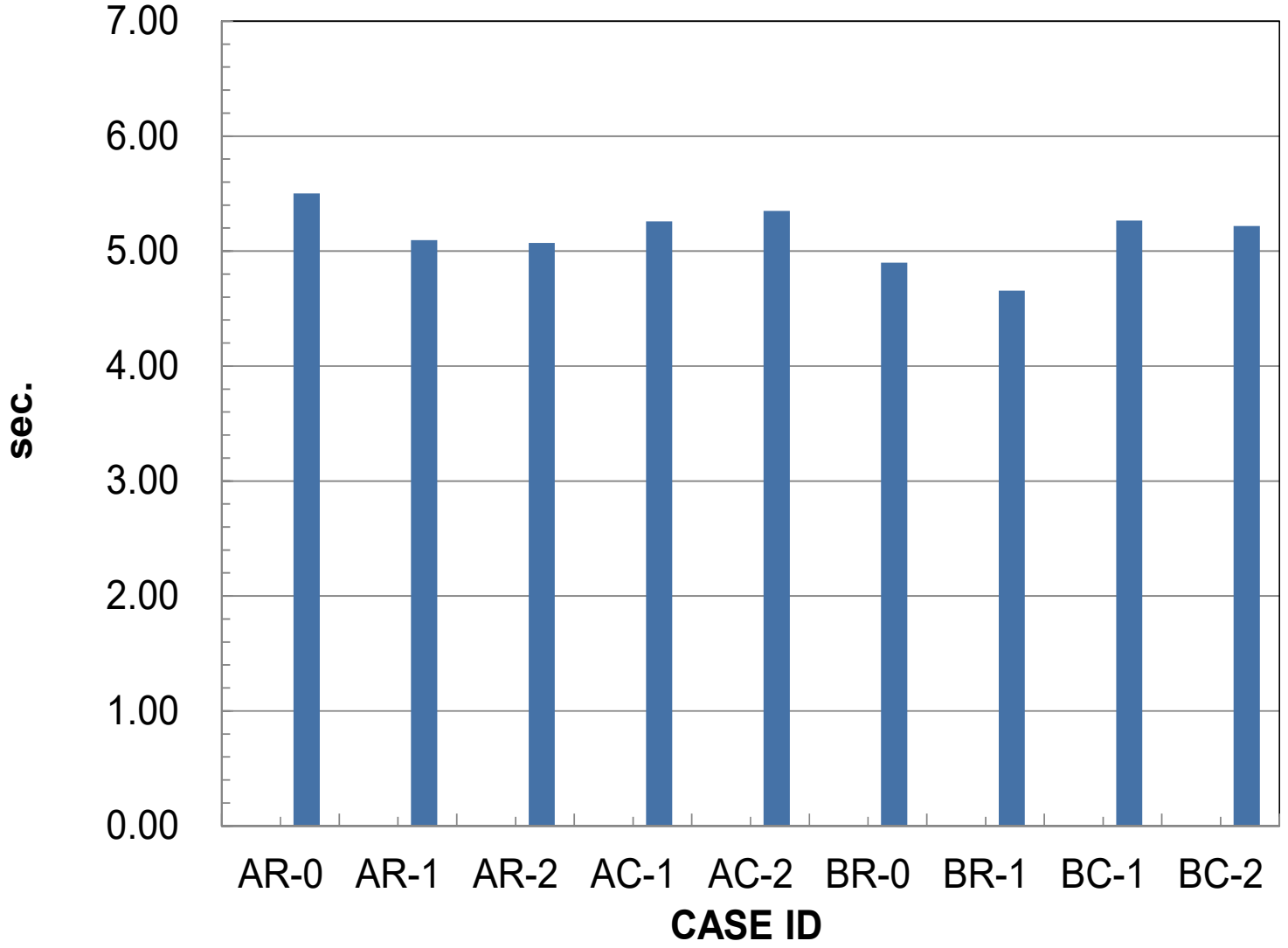




Results (Computation Time for Linear Solver): Down is Good !



Results: IvyB



FX10: 30 colors

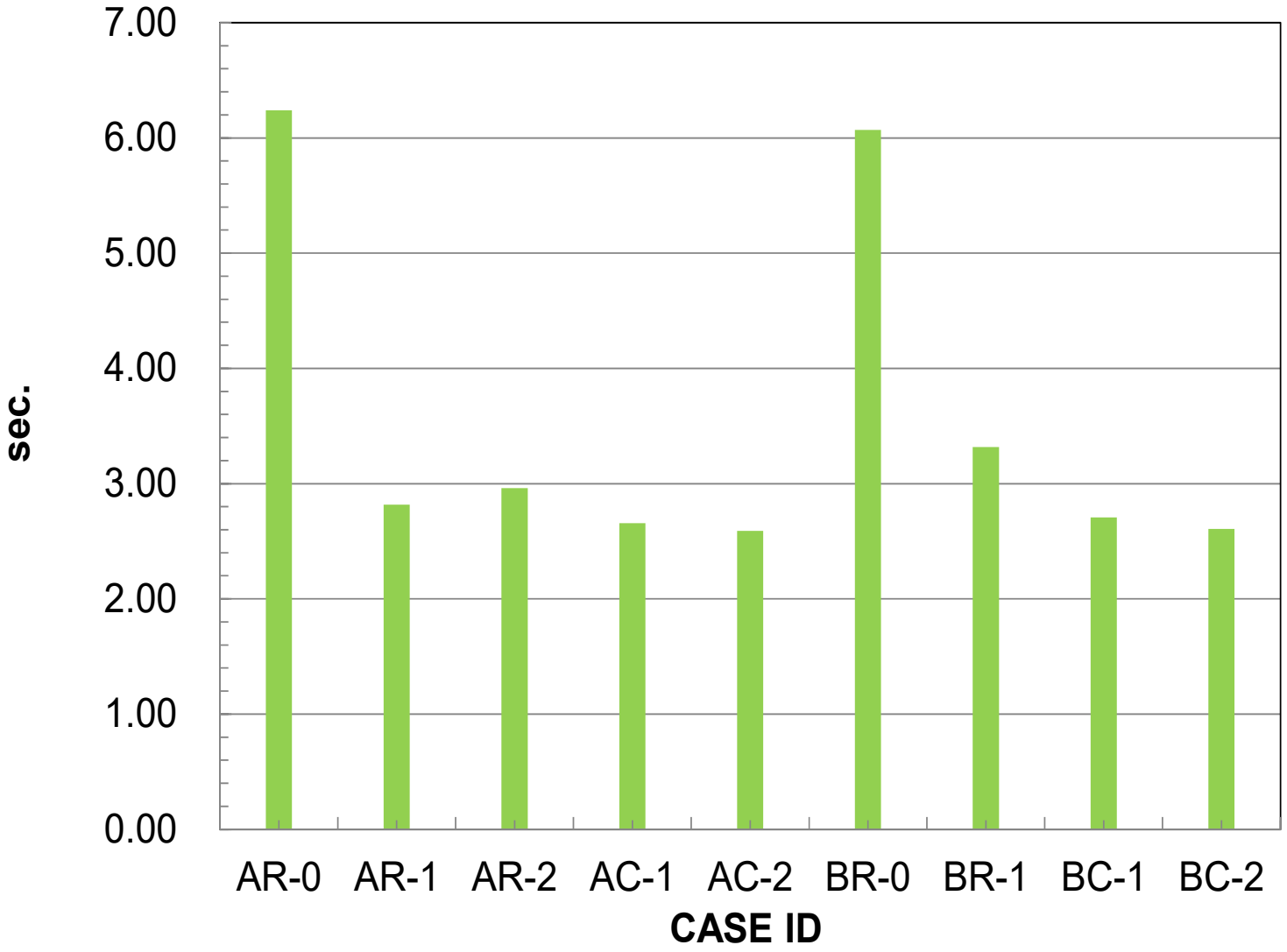
MIC: 10 colors

IvyB: 40 colors

IvyB

- Small effects of ELL
- Column-wise cases are worse

Results: MIC



FX10: 30 colors

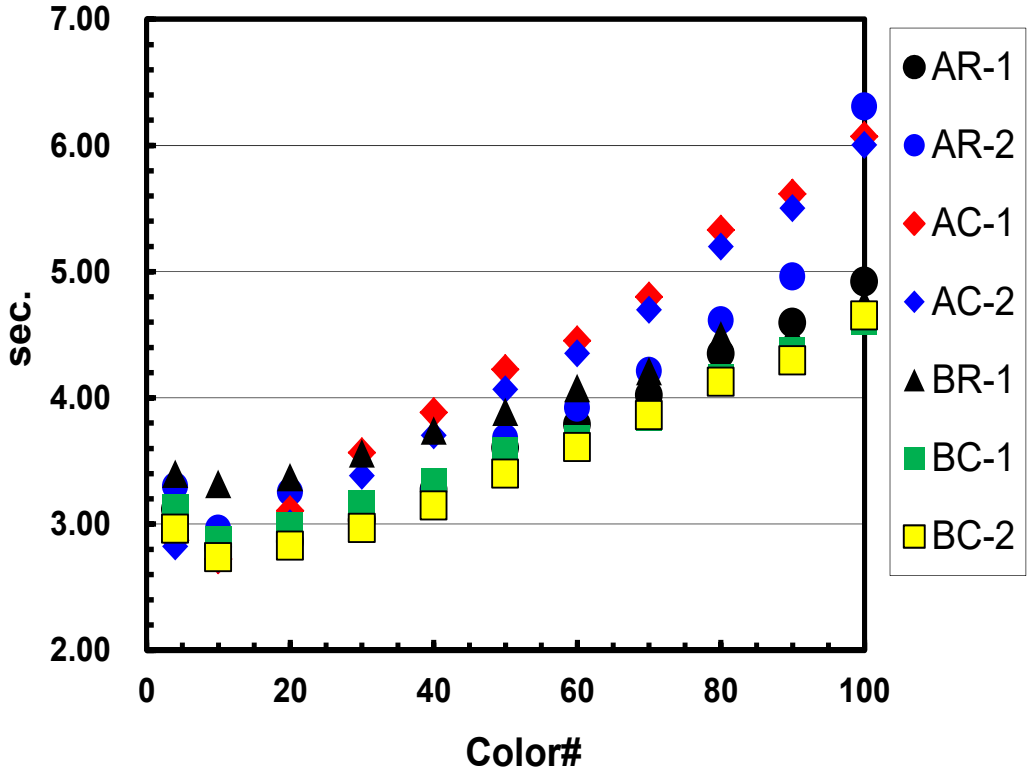
MIC: 10 colors

IvyB: 40 colors

MIC

- Effect of ELL is significant
- Column-wise cases (AC-X, BC-X) are better: Vectorization, SIMD
- Effect of blocking (XC-1=>XC-2)

Results: MIC



- Generally, BC-1 ■ (sequential) is better than AC-1 ◆(coalesced), if number of colors becomes larger
- "Sequential" can utilize cache more efficiently if number of color increases (this is not limited to MIC).
- AR-1 ●, BC-1 ■ , BC-2 ■ are rather robust
- Procedure with indirect accesses is slow with more than 80 colors.

- A: Coalesced
- B: Sequential

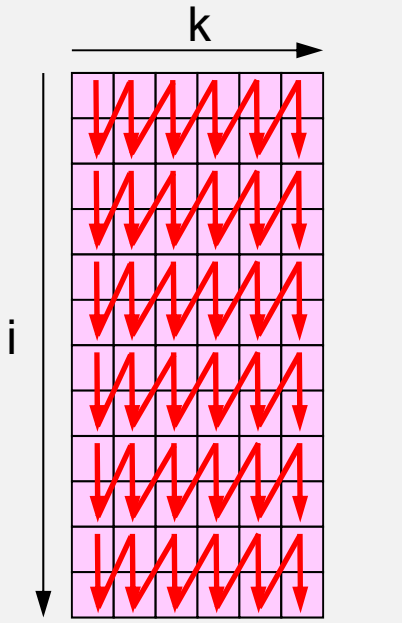
Compiler Options: Streaming-Cache/Prefetching

	AC-2		BC-2	
	sec.	GFLOPS	sec.	GFLOPS
-O3 -openmp -mmic -align array64byte (base)	2.654	10.53	**2.603**	**10.74**
-opt-streaming-stores always	3.165	8.832	3.159	8.849
-opt-streaming-cache-evict=0	2.625	10.65	2.600	10.75
-opt-streaming-cache-evict=1	2.639	10.59	2.605	10.73
-opt-streaming-stores always -opt-streaming-cache-evict=0	2.486	11.24	2.539	11.01
-opt-streaming-stores always -opt-streaming-cache-evict=1	2.477	11.29	2.556	10.94
-opt-streaming-stores always -opt-streaming-cache-evict=0 -opt-prefetch-distance=a,b	**2.385 (2,0)**	**11.72**	2.477 (8,1)	11.29
-opt-streaming-stores always -opt-streaming-cache-evict=1 -opt-prefetch-distance=c,d	2.404 (2,0)	11.63	2.487 (16,1)	11.24

ELL: Column-wise

Effect of “Streaming-Cache” is significant

```
!$omp parallel
    do icol= 1, NCOLORtot
!$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

Summary

- MIC
  - Effects of ELL are significant
  - Overhead by synchronization is larger if number of colors is larger
  - Column-wise outer loops are better
    - Effect of Streaming-Cache
    - Optimum pre-fetching distance
- Ivy Bridge
  - Effects of (ELL, configurations of outer loops, colors) are small
  - Out-of-Order

Programming Exercise

- Target: **`multicore/omp/src0`**
  - PCG with Diagonal Scaling Preconditioning
  - Apply “ELL” matrix storage format to the target

- Comparison of performance
- Analysis by profiler

OpenMP Version (C)

```
>$ cd
>$ cp /home/z30088/omp/omp-c0.tar .
>$ tar xvf omp-c0.tar
>$ cd multicore/omp/src0
>$ make

>$ ls ../run/sol0
   sol0

(modify go.sh (see below))
>$ pjsub go.sh
```

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:10:00"
#PJM -L "rscgrp=lecture9"
#PJM -g "gt99"
#PJM -j
#PJM -o “aaa.lst"
export OMP_NUM_THREADS=M      M=1,4 8,16

./sol0
```

OpenMP Version (Fortran)

```
>$ cd
>$ cp /home/z30088/omp/omp-f0.tar .
>$ tar xvf omp-f0.tar
>$ cd multicore/omp/src0
>$ make

>$ ls ../run/sol0
   sol0

(modify go.sh (see below))
>$ pjsub go.sh
```

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:10:00"
#PJM -L "rscgrp=lecture9"
#PJM -g "gt99"
#PJM -j
#PJM -o “aaa.lst"
export OMP_NUM_THREADS=M      M=1,4 8,16

./sol0
```

Original Program (not parallelized)

```
>$ cd <$O-TOP>

>$ cp /home/z30088/class_eps/ex.tar .

>$ tar xvf ex.tar

>$ cd ex

>$ cd C
>$ ls
     run    src

>$ cd ../F
>$ ls
     run    src

<$O-TOP>/F,<$O-TOP>/C: <$O-ex>
```

Procedures

- Code is parallelized, and reordered (omp-c0, omp-f0)
  - OpenMP
  - Reordered CM/RCM
- Apply ELL
  - poi_gen
  - solver_PCG

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)
```

Example: solver_PCG

Implementation of Mat-Vec (1/3)

```
      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 (2/3)

```
      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 (3/3)

```
      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
```

Report

- Deadline: 17:00 August 23 (Tue), 2016
  - 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
    - Numerical Experiments
    - Performance Analysis by Profiler
    - Remarks
  - Source list of the entire program (not included in the 20 pages above)