東京大学 THE UNIVERSITY OF TOKYO
東京大学情報基盤センター INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO
1, The University of Tokyo

Pascal vs KNL : Performance Evaluation with ICCG Solver

Tetsuya Hoshino[1], Satoshi Ohoshima[1], Toshihiro Hanawa[1], Kengo Nakajima[1], Akihiro Ida[1]

Motivation

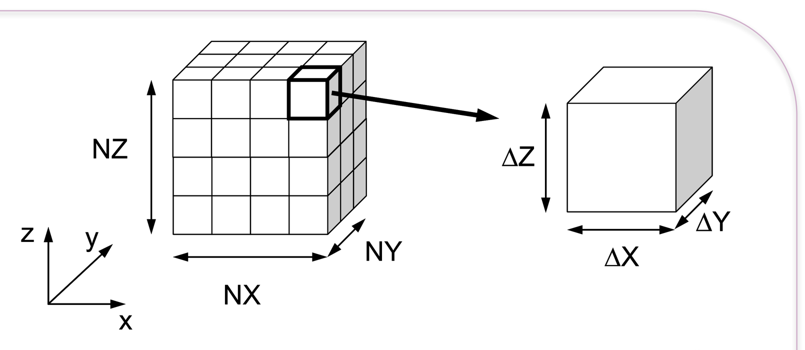
- Top 10 supercomputers of the Green 500 List use many-core devices
  - ✓ NVIDIA Pascal GPU, Intel Xeon Phi (KNL)
  - ✓ Development of the algorithms and optimization techniques for many-core devices is required.
- To know the characteristics, we evaluated the performance of new many-core devices

Reedbush (ITC, U-Toyo)
- Reedbush-U: CPU (Intel Xeon E5-2695) nodes x420
- Reedbush-H: CPU (Intel Xeon E5-2695) + GPU (NVIDIA Tesla P100) nodes x120

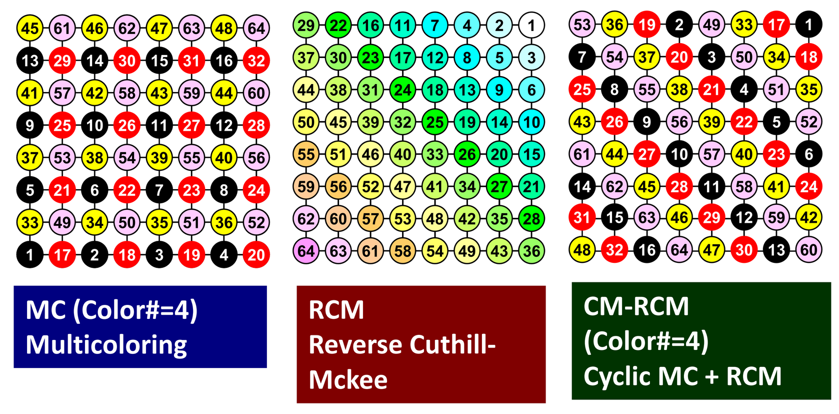
Oakforest-PACS (JCAHPC)
- TOP 500 #6 (#1 in Japan)
- 8,208 Intel Xeon Phi (KNL)
- 25 PF Peak Performance

ICCG Solver

- ICCG: Data Dependency for Incomplete Cholesky Factrization
- Fortran 90 + OpenMP/OpenACC
- Reordering required for parallelization
  - ✓ CM-RCM + Coalesced/Sequential
- Storage of Matrix
  - ✓ CRS, ELL(Ellpack-Itpack), SELL-C- σ



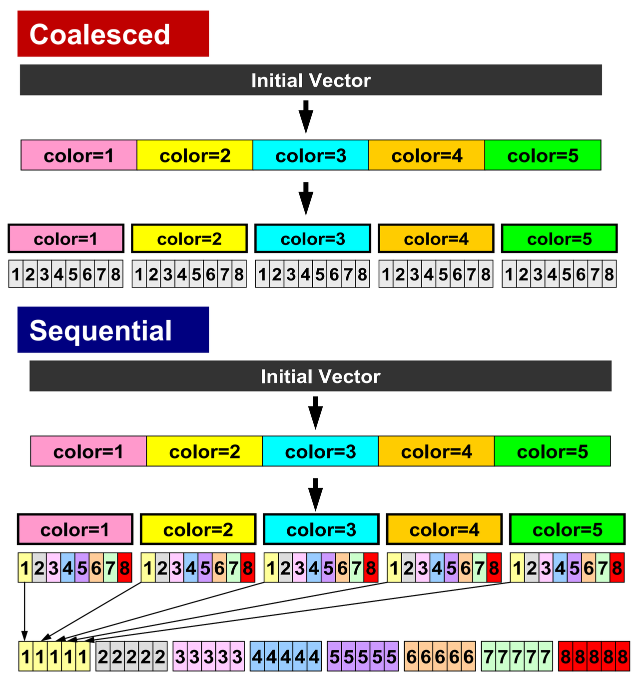
Target App: Finite Volume Method, Poisson Equations (128^3 cells)
- FDM-type mesh (7-pt. Stencil), Unstructured data structure
- SPD matrix



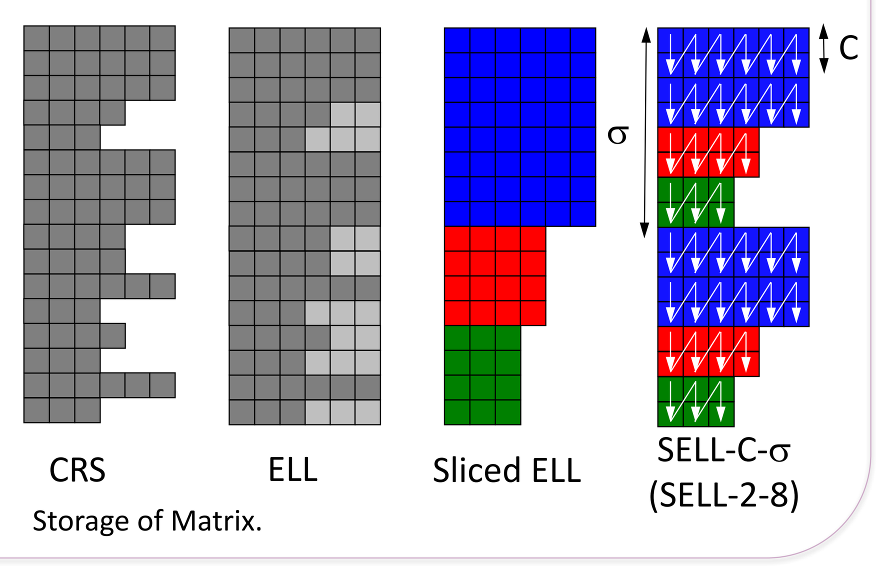
Coloring and reordering for parallelization. In this evaluations, we used CM-RCM reordering.

```
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
```

ICCG algorithm.



Coloring + Ordering (5 colors)



Storage of Matrix.

Performance Evaluation

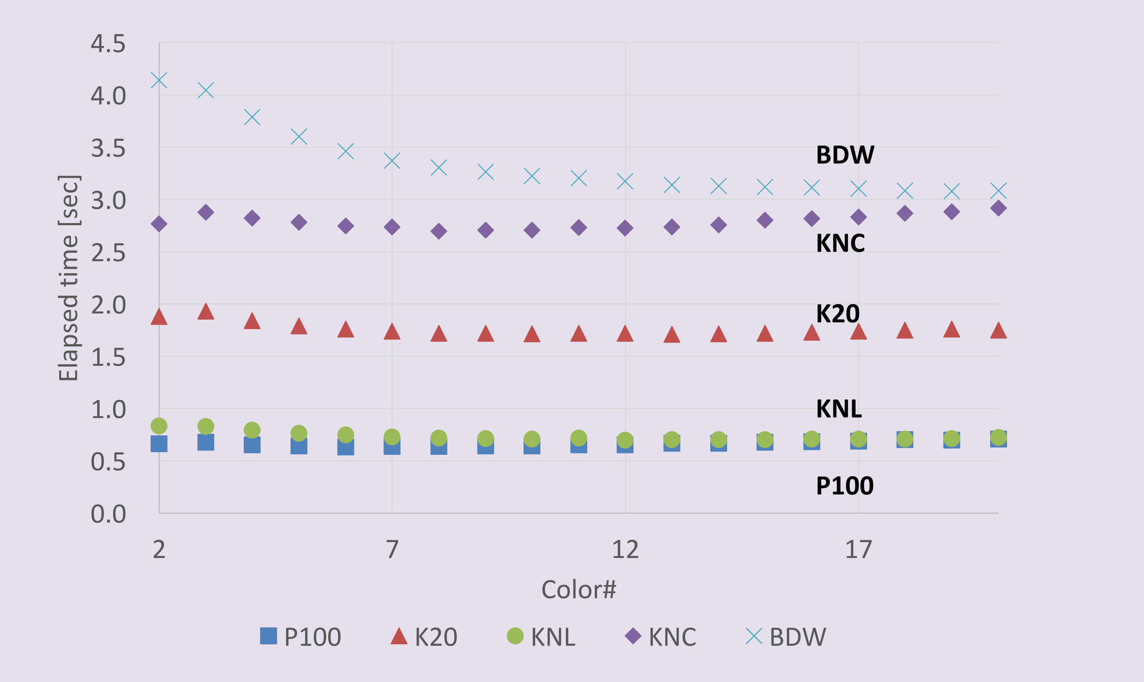
- Evaluate with several devices
  - ✓ GPU(Pascal/Kepler), Xeon Phi(KNC/KNL), CPU(Broadwell)
    - KNL: Flat-Quadrant, MCDRAM
  - ✓ Compiler
    - OpenACC: pgfortran 17.1 –acc –O3 -ta=tesla:cc60
    - OpenMP: ifort 17.0.1 –align array64byte –O3 –xMIC-AVX512 -qopenmp –qoptstreaming-stores=always –qopt-streaming-cache-evict=0

	GPU P100	GPU K20	Xeon Phi KNL	Xeon Phi KNC	Broadwell
Frequency (GHz)	1.480	0.732	1.40	1.053	2.10
Core# (Max thread#)	1,792	896	68 (272)	60 (240)	18 (18)
Peak Performance (GFLOPS)	5,304	1,311	3,046	1,010	604
Memory (GB)	16	6	16	8	128
Bandwidth (GB/sec., Stream Triad)	**534**	**179**	**490**	**159**	**65.5**
System	Reedbush-H	TSUBAME2.5	Oakforest-PACS	KNSC	Reedbush-U

Evaluation cases

	Numbering	Storage of Matrix
c-CRS	Coalesced	CRS
c-Sliced-ELL		Sliced-ELL (wit blockin)
c-SELL-C- σ		SELL-C- σ
s-CRS	Sequential	CRS
s-Sliced-ELL		Sliced-ELL (with blocking)
s-SELL-C- σ		SELL-C- σ

Performance of the baseline version



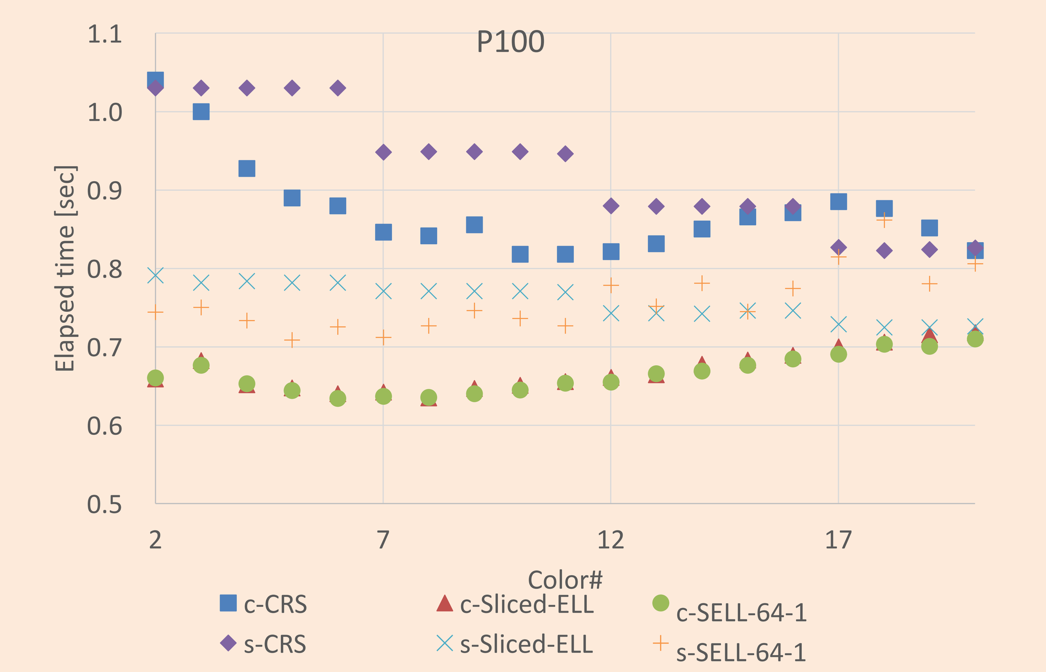
- Optimizations for P100
1. Baseline
   - ✓ Insert !$acc kernels to all loops to be parallelized
2. Async
   - ✓ Attach async(0) clause
3. Thread
   - ✓ Optimize gang/vector parameters
4. Fusion
   - ✓ Kernel fusion to reduce kernel call cost

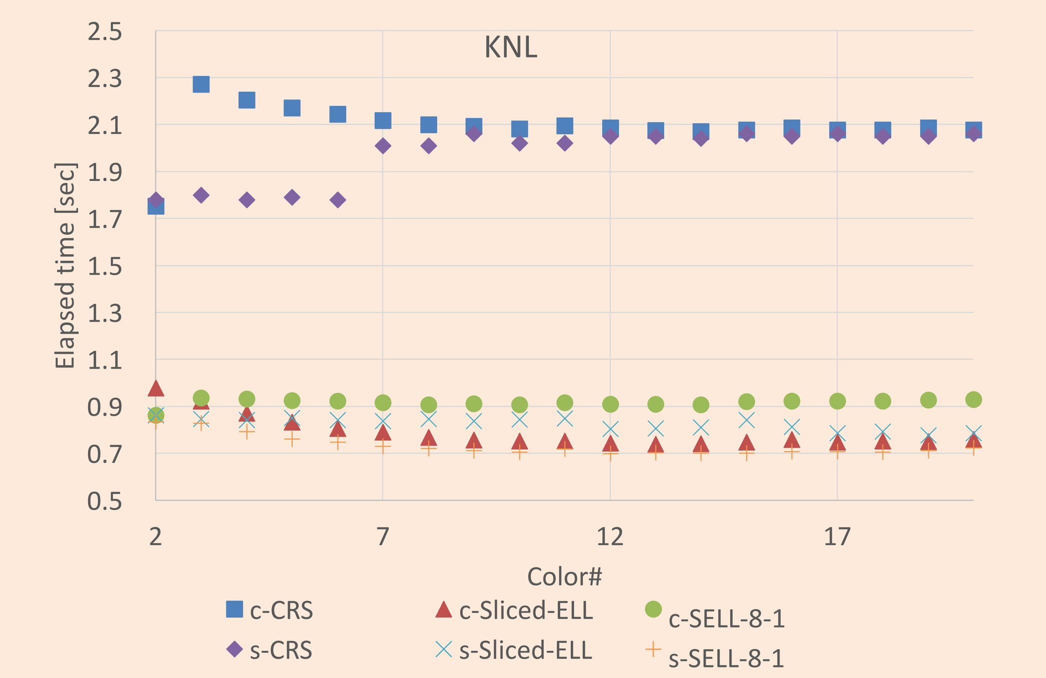
```
!$omp parallel do private(... )
!$acc kernels
!$acc loop independent gang
   do ip= 1, PEsmpTOT
     ip1= (ip-1)*NCOLORtot + ic
!$omp simd
!$acc loop independent vector
     do i= index(ip1-1)+1, index(ip1)
       …
     enddo
   enddo
!$omp end parallel do
!$acc end kernels
```

Baseline Implementation of OpenMP/OpenACC

- Optimizations for KNL
1. Baseline
   - ✓ Insert !$acc parallel do to all loops to be parallelized
2. mvparallel 1
   - ✓ Move !$acc parallel to outside of color loop
3. nowait
   - ✓ Use !$omp end do nowait
4. mvparallel 2
   - ✓ Move !$parallel to outside of convergence loop
5. rmompdo
   - ✓ Parallelize loops by hand without using !$omp do
6. rmreduction
   - ✓ Reduction by hand
7. loopscheduling
   - ✓ Parallelize loops by using application knowledge

Evaluation of the Numbering and Storage of Matrix (Baseline version)





Evaluation of the optimizations

- Reducing synchronization costs is important
  - ✓ Attach async clause, which reduces the kernel call overhead, is most effective for P100
  - ✓ A series of optimization, which reduces synchronization, is effective for KNL