2019 Intel® eXtreme Performance Users Group (IXPUG) Annual Conference

Cost-Efficiency of Large-Scale Electronic Structure Simulations with Intel Xeon Phi Processors

Hoon Ryu, Ph.D.

(E: elec1020@kisti.re.kr)

Division of National Supercomputing R&D
Korea Institute of Science and Technology Information (KISTI)

KiSTi 한국과학기술정보연구원 Korea Institute of Science and Technology Information www.kisti.re.kr

국가슈퍼컴퓨팅연구소 National Institute of Supercomputing and Networking

IPCC KISTI

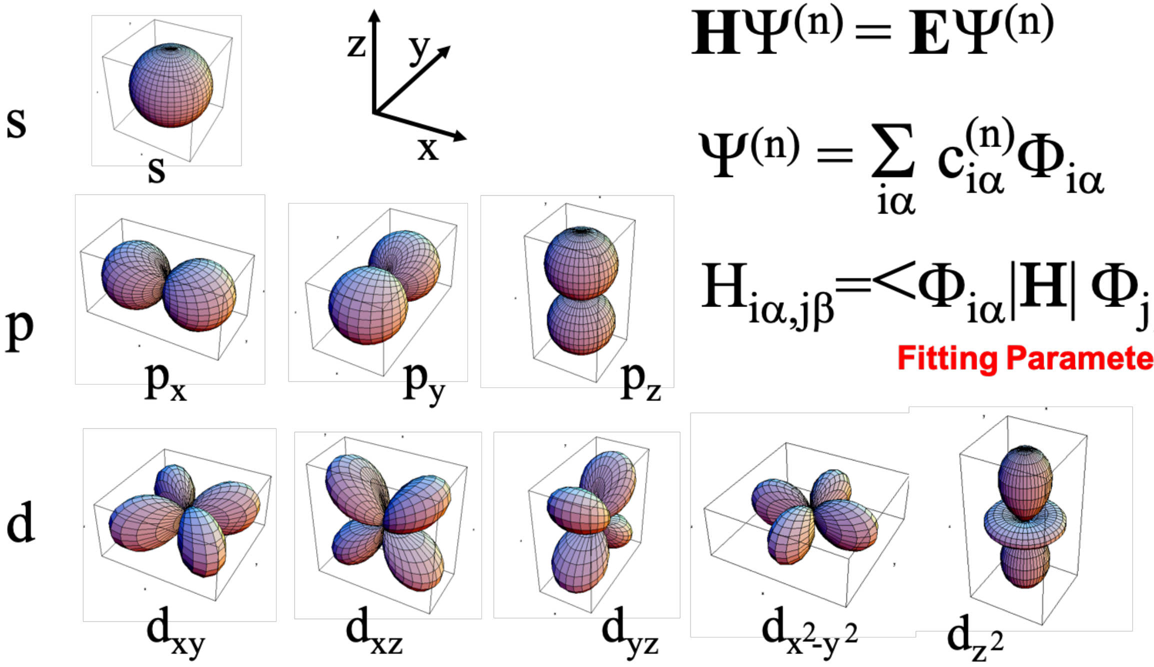
Tight-binding (TB) Theory

An empirical approach: short intro & cons/pros.

Basis: $\Phi_{i\alpha}$

i = atom label

α = basis state



$$\mathbf{H}\Psi^{(n)} = \mathbf{E}\Psi^{(n)}$$

$$\Psi^{(n)} = \sum_{i\alpha} c_{i\alpha}^{(n)} \Phi_{i\alpha}$$

$$H_{i\alpha,j\beta} = <\Phi_{i\alpha}|H|\,\Phi_{j\beta}>$$

Fitting Parameters

P. Vogl *et al.*, *J. of Phys. Chem. Solids* **44**, 5, 365-378 (1983)

G. Klimeck *et al.*, IEEE Trans. Elec. Dev. **54**, 9, 2079-2089 (2007)

Empirical Tight-binding (TB) Theory

- Finite sets of localized, orthogonal basis
- Mesh: artificial or the one representing crystalline
- s (1), sp3 (4), sp3d5s* (10), …..
  → Size doubles for considerations of S.O.
- Will it be possible to describe confined structures with a finite number of plane-wave basis?
  → # of basis needed to describe a non-periodic function with a (Discrete) Fourier Transform?

Cons and Pros

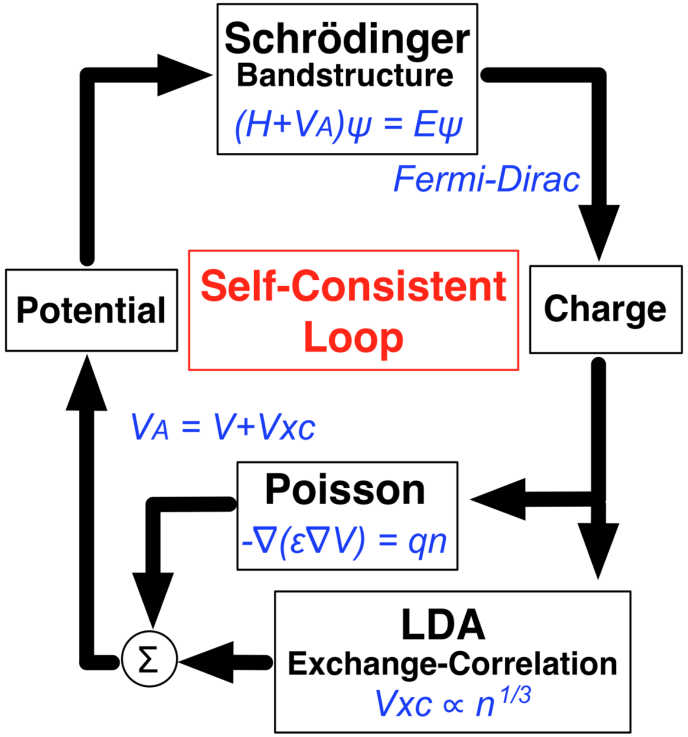
- Major numerical cost happens in diagonalizations
  → N-dimension integrals?
- Not a first principal theory
  → Params fit to reproduce bandstructures known through other methods

TB Electronic Structure Calculations

In a perspective of “numerical analysis”

- Two PDE-coupled Loop: **<u>Schrödinger Equation</u>** and **<u>Poisson Equation</u>**
- Both equations involve **<u>system matrices</u>** (Hamiltonian and Poisson)

  → DOFs of those matrices are proportional to the # of grids in the simulation domains



- (Stationary) Schrödinger Equations

  → **<u>Normal Eigenvalue Problem</u>** $H\Psi = E\Psi$

- Poisson Equations

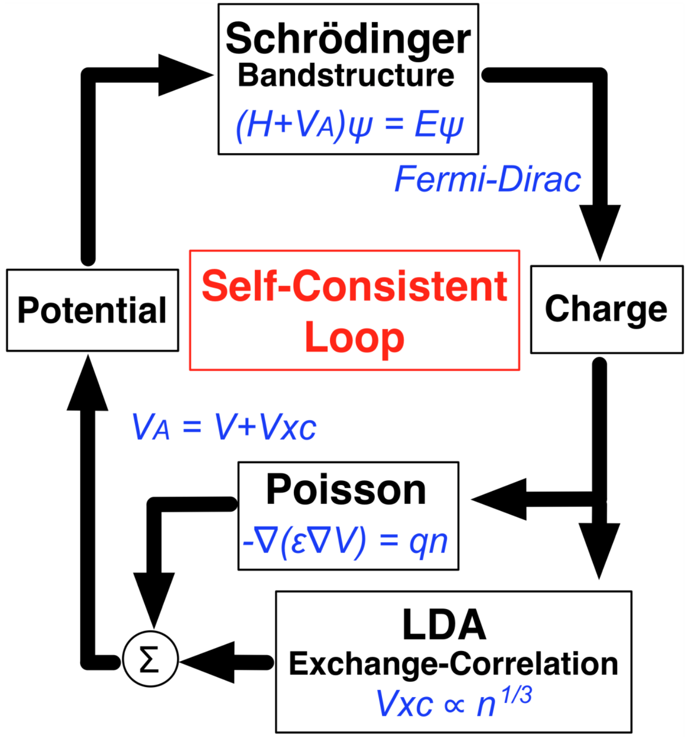
  → **<u>Linear System Problem</u>** $-\nabla(\varepsilon\nabla V) = \rho$ → $Ax = b$

TB Electronic Structure Calculations

In a perspective of “numerical analysis”

- Two PDE-coupled Loop: **Schrödinger Equation** and **Poisson Equation**
- Both equation involve **system matrices** (Hamiltonian and Poisson)

→ DOFs of those matrices are proportional to the # of grids in the simulation domains



- Schrödinger Equations

→ **Normal Eigenvalue Problem** $H\Psi = E\Psi$

- Poisson Equations

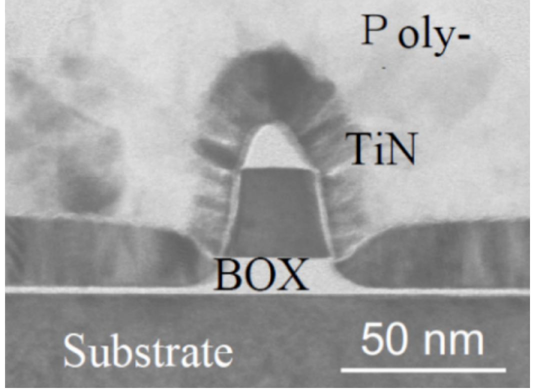
→ **Linear System Problem** $-\nabla(\varepsilon\nabla V) = \rho \rightarrow Ax = b$

How large are these system matrices?
Why do we need to handle those?

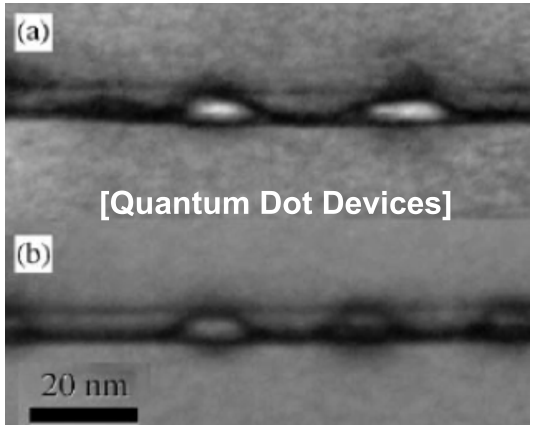
Needs for "Large" Electronic Structures

Electron motion happens in cores, but we need more

1. Quantum Simulations of "Realizable" Nanoscale Materials and Devices
→ Needs to handle large-scale atomic systems (~ A few tens of nms)



[Logic Transistors (FinFET)]



[Quantum Dot Devices]

30nm^3 Silicon Box? → About million atoms

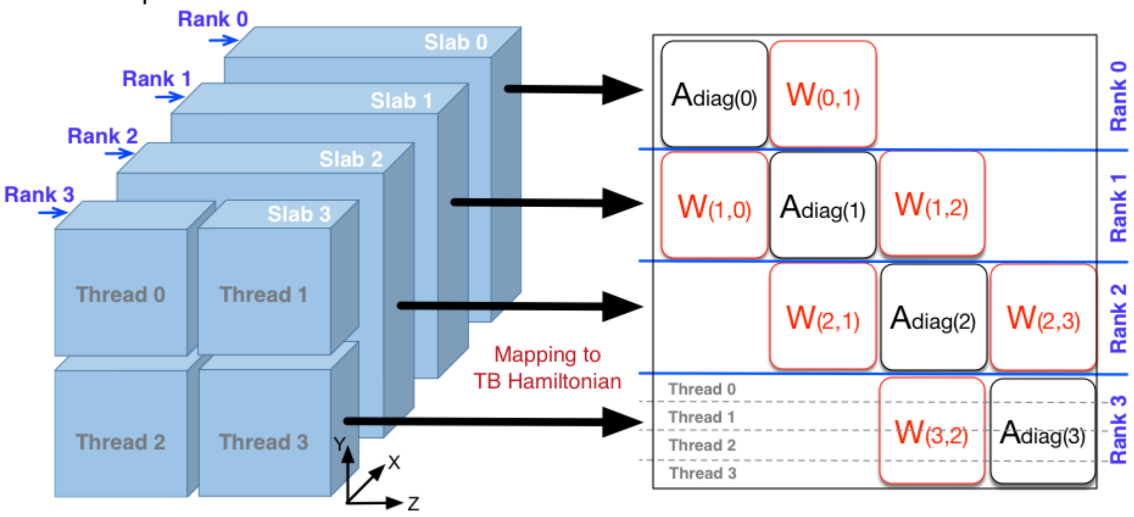
2. DOF of Matrices of Governing Equations
→ Linearly proportional to # of atoms (w/ some weight)

3. Parallel Computing

$$Ax = b$$

$$H\Psi = E\Psi$$

Decomposition of Simulation Domains



Development Strategy: DD, Matrix Handling

System matrices for Schrödinger and Poisson equations

Schrödinger Equation

- Normal Eigenvalue Problem (Electronic Structure)
- Hamiltonian is always symmetric

$$\boxed{H\Psi = E\Psi}$$

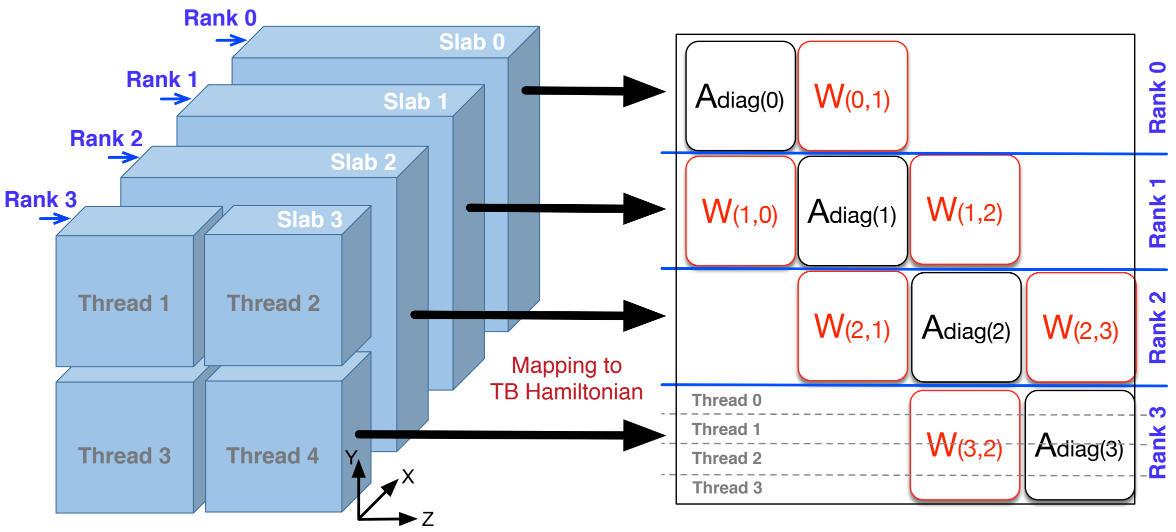
Poisson Equation

- Linear System Problem (Electrostatics: Q-V)
- Poisson matrix is always symmetric

$$\boxed{-\nabla(\varepsilon\nabla V) = \rho}$$

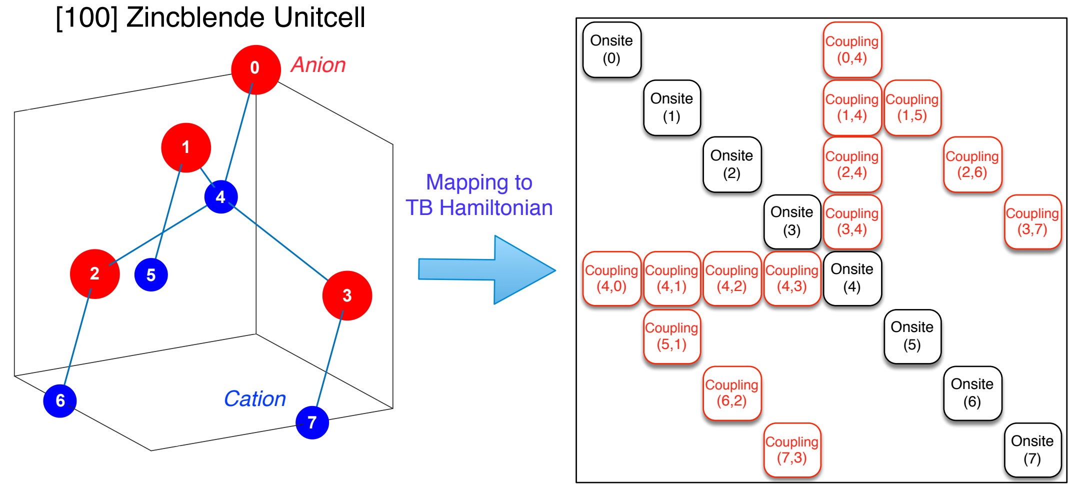
Domain Decomposition

- MPI + OpenMP
- Effectively multi-dimensional decomposition



Matrix Handling

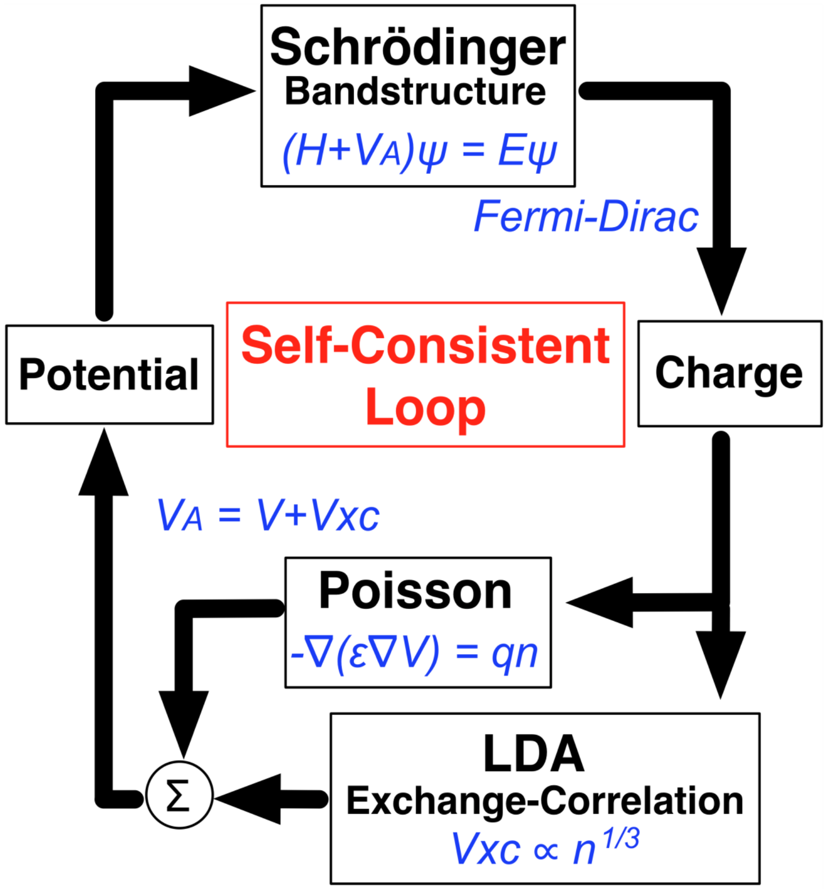
- Tight-binding Hamiltonian (Schrödinger Eq.)
- Finite Difference Method (Poisson Eq.)
- Nearest Neighbor Coupling: Highly Sparse → CSR



Development Strategy: Numerical Algorithms

Schrödinger equations

Self-consistent Loop for Device Simulations



Schrödinger Bandstructure

$(H+V_A)\psi = E\psi$

Fermi-Dirac

Charge

Poisson

$-\nabla(\varepsilon\nabla V) = qn$

LDA Exchange-Correlation

$V_{xc} \propto n^{1/3}$

Σ

$V_A = V+V_{xc}$

Potential

Self-Consistent Loop

Schrödinger Eqs. w/ LANCZOS Algorithm

→ C. Lanczos, *J. Res. Natl. Bur. Stand.* 45, 255

- Normal Eigenvalue Problem (Electronic Structure)
- Hamiltonian is always symmetric
- Original Matrix → T matrix-reduction
- Steps for Iteration: Purely Scalable Algebraic Ops.

$$\boxed{H\Psi = E\Psi}$$

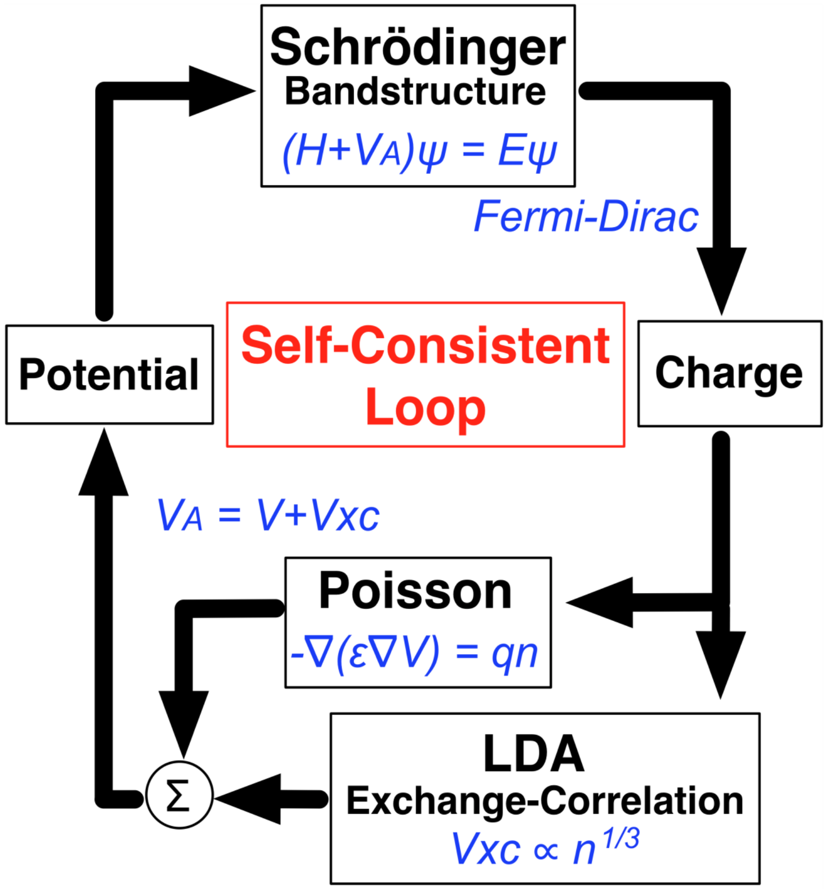
```
$\mathbf{v}_i$: (Nx1) vectors (i = 0, …, $\mathbf{K}$); $\mathbf{a}_i$ and $\mathbf{b}_i$: scalars (i = 1, …, $\mathbf{K}$)
$\mathbf{v}_0 \leftarrow \mathbf{0}$, $\mathbf{v}_1$ = random vector with norm 1 ;
$\mathbf{b}_1 \leftarrow \mathbf{0}$ ;
loop for (j=1; j<=$\mathbf{K}$ ; j++)
  $\mathbf{w}_j \leftarrow \mathbf{A}\mathbf{v}_j$ ;
  $\mathbf{a}_j \leftarrow \mathbf{w}_j \bullet \mathbf{v}_j$ ;
  $\mathbf{w}_j \leftarrow \mathbf{w}_j - \mathbf{a}_j\mathbf{v}_j - \mathbf{b}_j\mathbf{v}_{j-1}$ ;
  $\mathbf{b}_{j+1} \leftarrow \|\mathbf{w}_j\|$ ;
  $\mathbf{v}_{j+1} \leftarrow \mathbf{w}_j / \mathbf{b}_{j+1}$ ;
  construct T matrix;
end loop
```

$$\Longrightarrow \quad T = \begin{pmatrix} a_1 & b_2 & 0 & \cdots & \cdots & 0 \\ b_2 & a_2 & b_3 & & & \vdots \\ 0 & b_3 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & b_{k-1} & 0 \\ \vdots & & & b_{k-1} & a_{k-1} & b_k \\ 0 & \cdots & \cdots & 0 & b_k & a_k \end{pmatrix}$$

Development Strategy: Numerical Algorithms

Poisson equations

Self-consistent Loop for Device Simulations

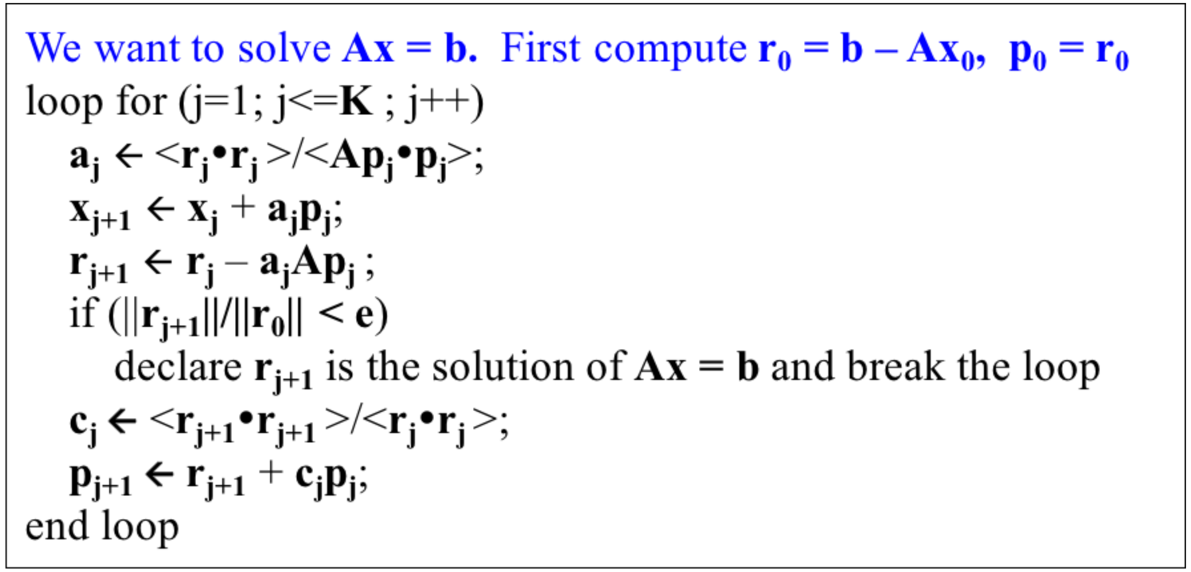


Poisson Eqs. w/ CG Algorithm

→ Hestenes et al., *J. Res. Natl. Bur. Stand.* 49, 409

- Conv. Guaranteed: Symmetric & Positive Definite
- Poisson is always S & PD.
- Steps for Iteration: Purely Scalable Algebraic Ops.

$$-\nabla(\varepsilon\nabla V)=\rho$$

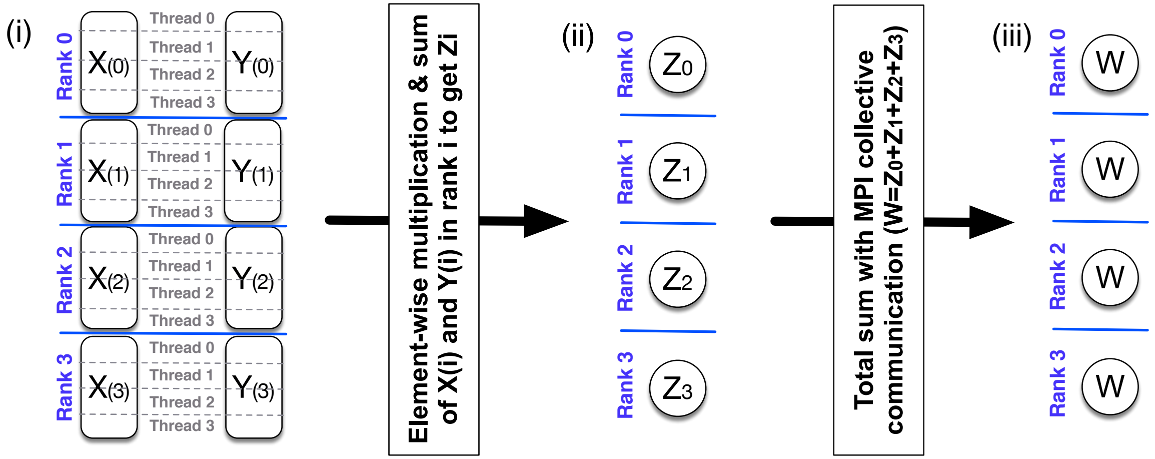


```
We want to solve $\mathbf{Ax} = \mathbf{b}$.  First compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$,  $\mathbf{p}_0 = \mathbf{r}_0$
loop for (j=1; j<=K ; j++)
    $\mathbf{a}_j \leftarrow <\mathbf{r}_j \bullet \mathbf{r}_j>/<\mathbf{Ap}_j \bullet \mathbf{p}_j>$;
    $\mathbf{x}_{j+1} \leftarrow \mathbf{x}_j + \mathbf{a}_j\mathbf{p}_j$;
    $\mathbf{r}_{j+1} \leftarrow \mathbf{r}_j - \mathbf{a}_j\mathbf{Ap}_j$ ;
    if ($||\mathbf{r}_{j+1}||/||\mathbf{r}_0|| < \mathbf{e}$)
        declare $\mathbf{r}_{j+1}$ is the solution of $\mathbf{Ax} = \mathbf{b}$ and break the loop
    $\mathbf{c}_j \leftarrow <\mathbf{r}_{j+1} \bullet \mathbf{r}_{j+1}>/<\mathbf{r}_j \bullet \mathbf{r}_j>$;
    $\mathbf{p}_{j+1} \leftarrow \mathbf{r}_{j+1} + \mathbf{c}_j\mathbf{p}_j$;
end loop
```

Performance Bottleneck?

Matrix-vector multiplier: Sparse matrices

Vector Dot-Product (VVDot)



(i)
Rank 0
Rank 1
Rank 2
Rank 3
X(0)
X(1)
X(2)
X(3)
Y(0)
Y(1)
Y(2)
Y(3)
Thread 0
Thread 1
Thread 2
Thread 3
Element-wise multiplication & sum of X(i) and Y(i) in rank i to get Zi
(ii)
Z0
Z1
Z2
Z3
Total sum with MPI collective communication (W=Z0+Z1+Z2+Z3)
(iii)
W
W
W
W

Main Concerns for Performance

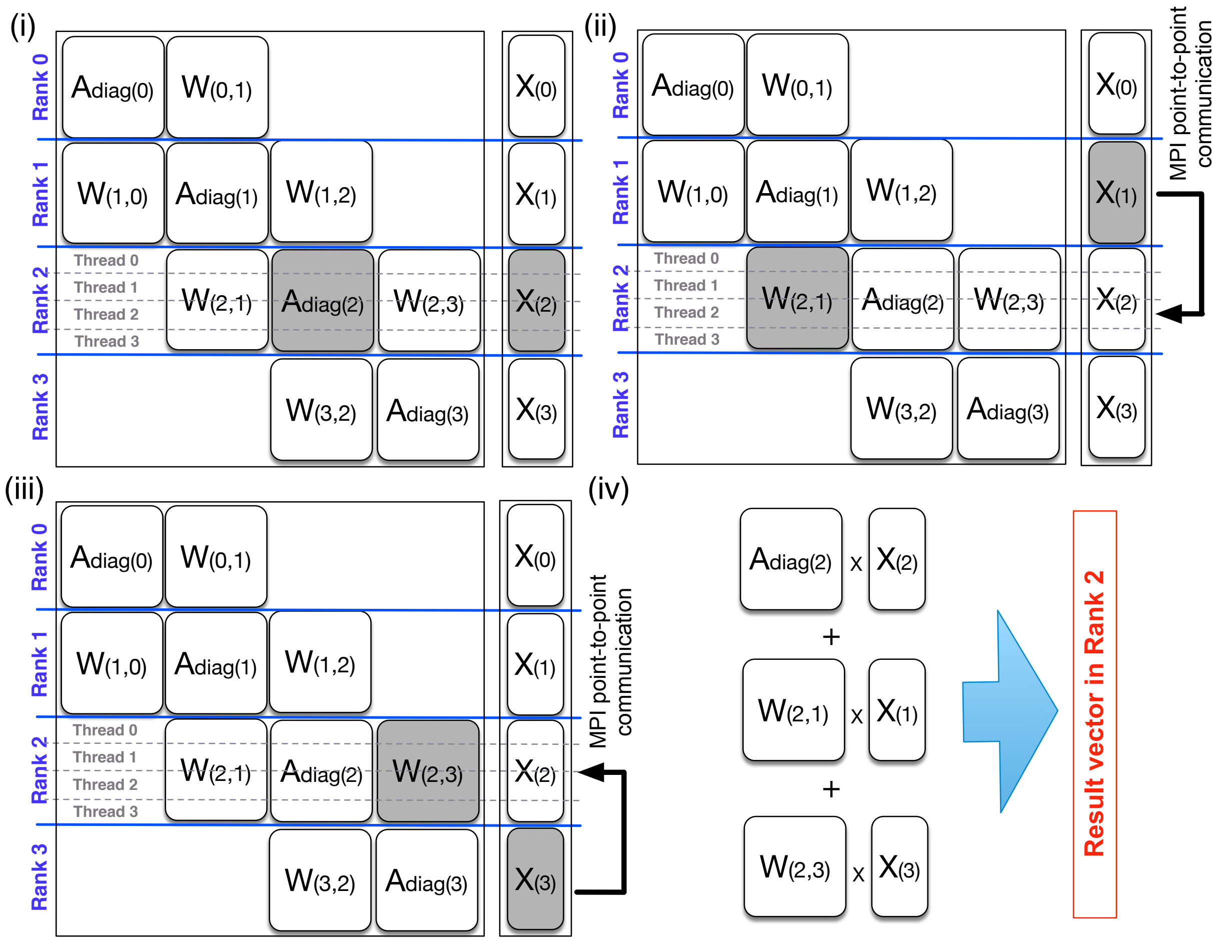
- Collective Communication

  → May not be the main bottleneck as we only need to collect a single value from a single MPI process
- Matrix-vector Multiplier

  → Communication happens, but would not be a critical problem as it only happens between adjacent ranks

  → Data locality affects vectorization efficiency

(Sparse) Matrix-vector Multiplier (MVMul)



Performance w/ Intel® KNL Processors

Single-node Performance: Whole domain in MCDRAM

Description of BMT Target and Test Mode

- 10 CB states in 16x43x43(nm^3) [100] Si:P quantum dot
  → Material candidates for Si Quantum Info. Processors (Nature Nanotech. **9**, 430)
  → 15.36Mx15.36M Hamiltonian Matrix (~11GB)
- Xeon Phi 7210: 64 cores
- MCDRAM control w/ **numactrl**; Quad Mode

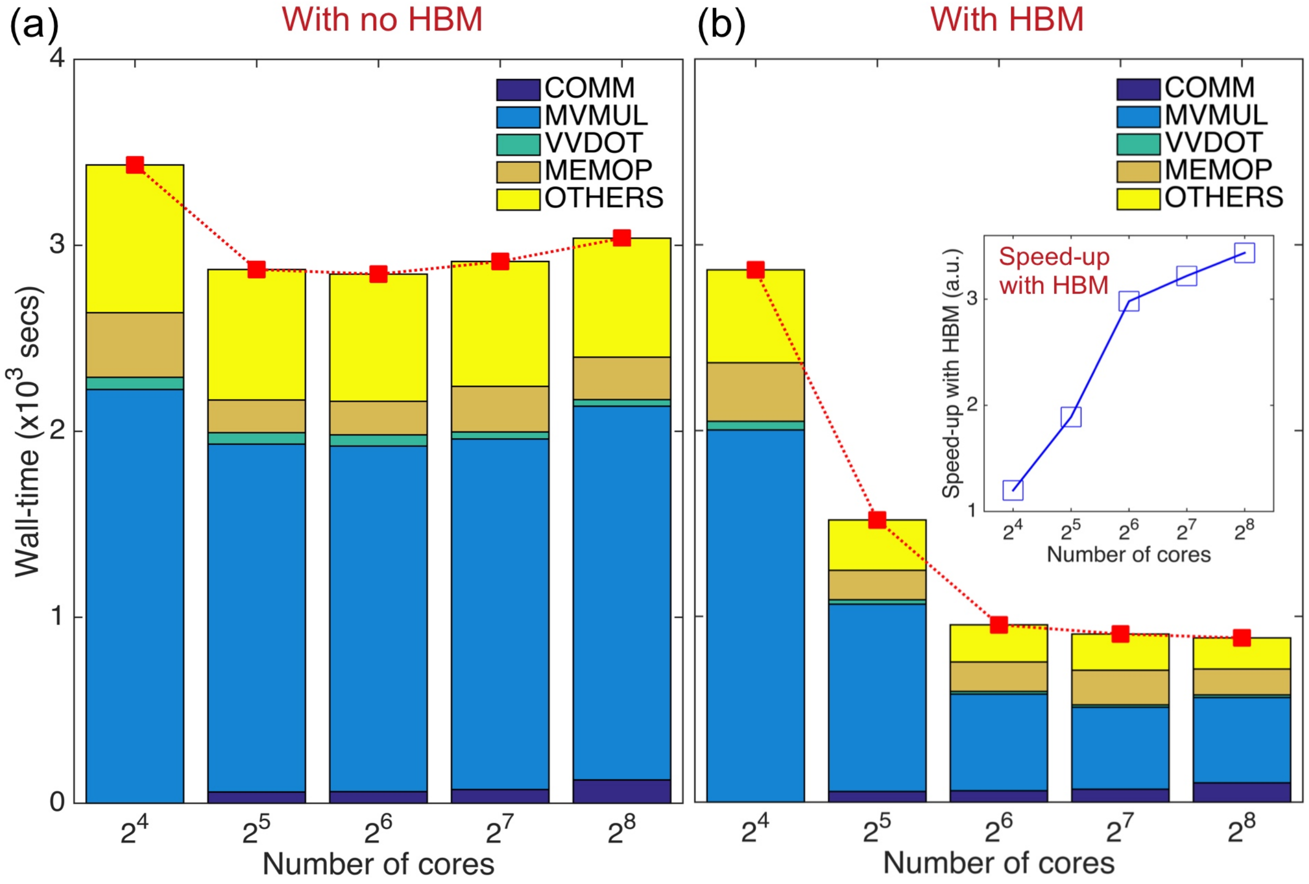
Results

To be presented in IEEE CLUSTER (2019)

- With no MCDRAM
  → No clear speed-up beyond 64 cores
- **With MCDRAM**
  → Up to ~4x speed-up w.r.t. the case w/ no MCDRAM
  → Intra-node scalability up to 256 cores

Points of Questions

- How is the performance compared to the one under other computing environments? (GPU, CPU-only etc..)
  → In terms of speed and **energy consumption**



2^4 cores = (1 MPI proc(s), 16 threads), 2^7 cores = (2 MPI proc(s), 64 threads)
2^5 cores = (2 MPI proc(s), 16 threads), 2^8 cores = (4 MPI proc(s), 64 threads)
2^6 cores = (2 MPI proc(s), 32 threads)

Performance w/ Intel® KNL Processors

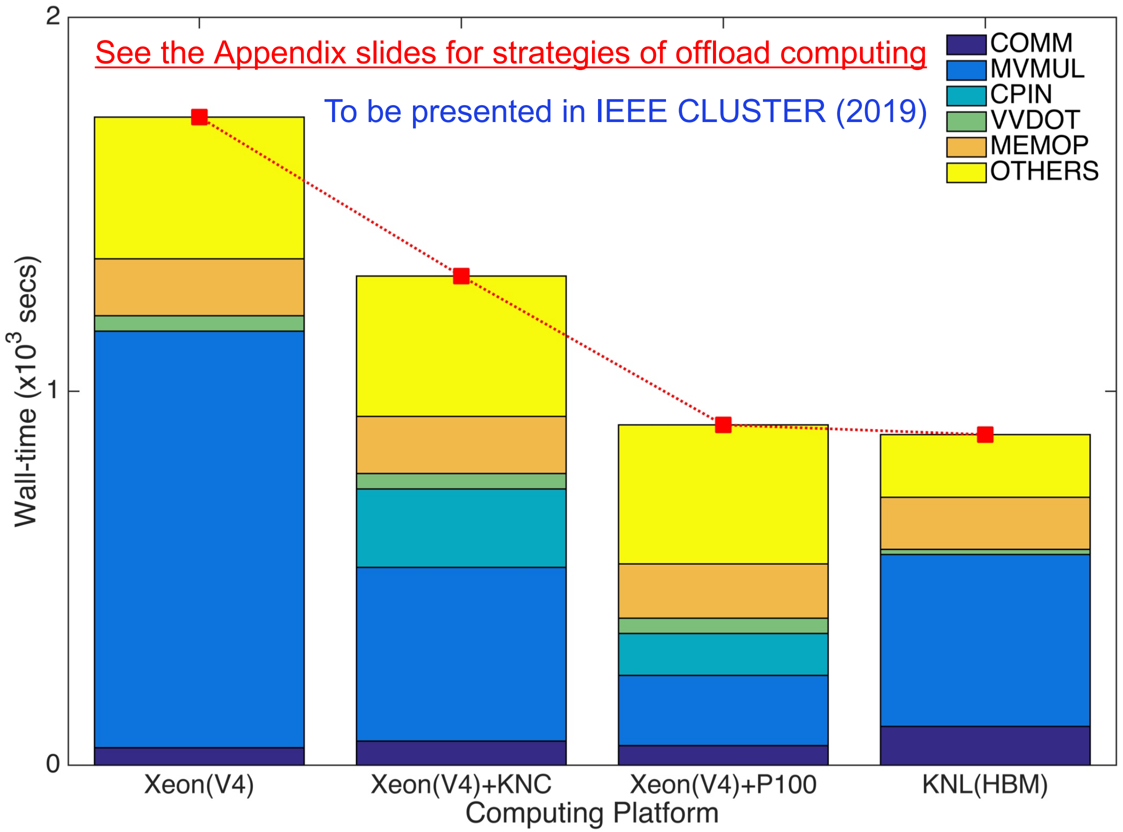
Speed in various computing platforms

Description of BMT Target and Test Mode

- 10 CB states in 16x43x43(nm^3) [100] Si:P quantum dot
  → Material candidates for Si Quantum Info. Processors (Nature Nanotech. **9**, 430)
  → 15.36Mx15.36M Hamiltonian Matrix (~11GB)
- Specs of Other Platforms
  → Xeon(V4): 24 cores of Broadwell (BW) 2.50GHz
  → Xeon(V4)+KNC: 24 cores BW + 2 KNC 7120 cards
  → Xeon(V4)+P100: 24 cores BW + 2 P100 cards
  → KNL(HBM): the one described so far

Results

- **KNL slightly beats Xeon(V4)+P100**
  → Copy-time (CPIN): a critical bottleneck of PCI-E devices
  → P100 shows better kernel speed, but the overall benefit reduces due to data-transfer between host and devices
  → CPIN would even increase if we consider periodic BCs
- **Another critical figure of merit: Energy-efficiency**



See the Appendix slides for strategies of offload computing

To be presented in IEEE CLUSTER (2019)

Xeon(V4) = (2 MPI proc(s), 12 threads)
Xeon(V4) + KNC = (2 MPI proc(s), 12 threads) + 2 KNC 7120 cards
Xeon(V4) + P100 = (2 MPI proc(s), 12 threads) + 2 P100 cards
KNL (HBM) = (4 MPI proc(s), 64 threads) with HBM

Performance w/ Intel® KNL Processors

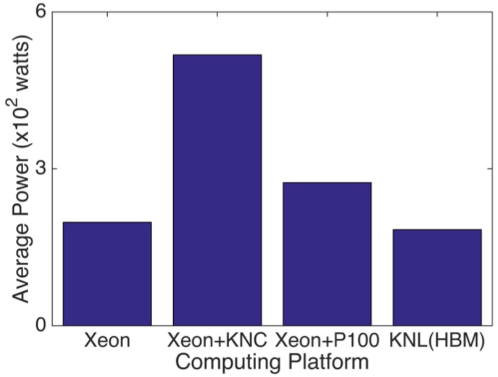
Energy consumption in various computing platform

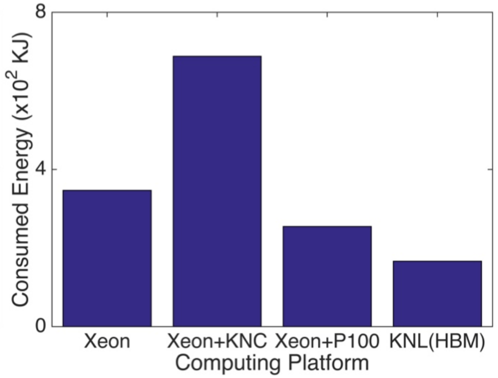
Description of BMT Target and Test Mode

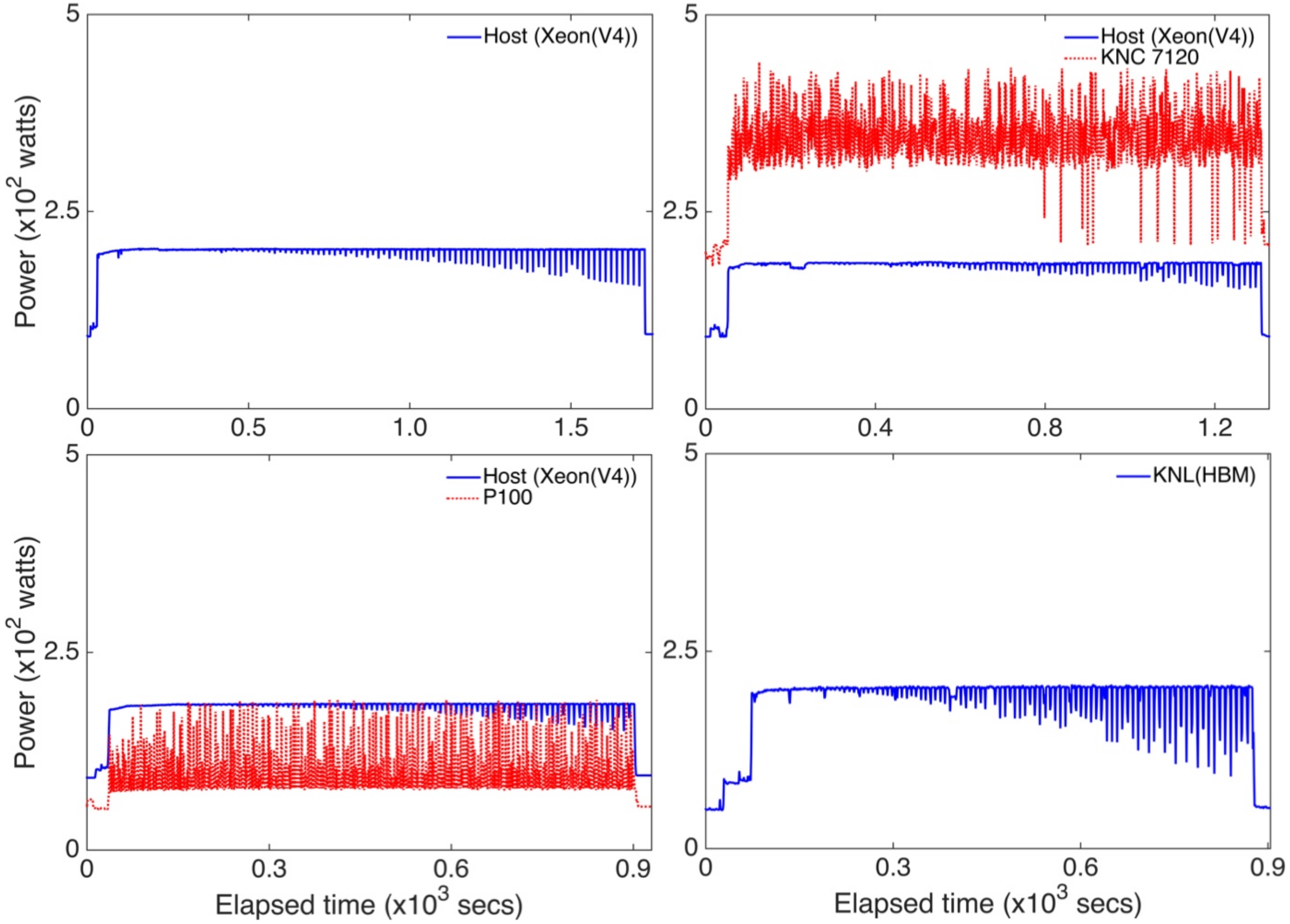
- 10 CB states in 16x43x43(nm^3) [100] Si:P quantum dot
  → Hamiltonian DOF: 15.36Mx15.36M (~11GB)
- Description of Device Categories
  → Xeon(V4): 24 cores of Broadwell (BW) 2.50GHz
  → Xeon(V4)+KNC: 24 cores BW + 2 KNC 7120 cards
  → Xeon(V4)+P100: 24 cores BW + 2 P100 cards
  → KNL(HBM): the one described so far

Power Measurement

- w/ RAPL (Running Ave. Power Limit) API
- Host (CPU+Memory), PCI-E Devices





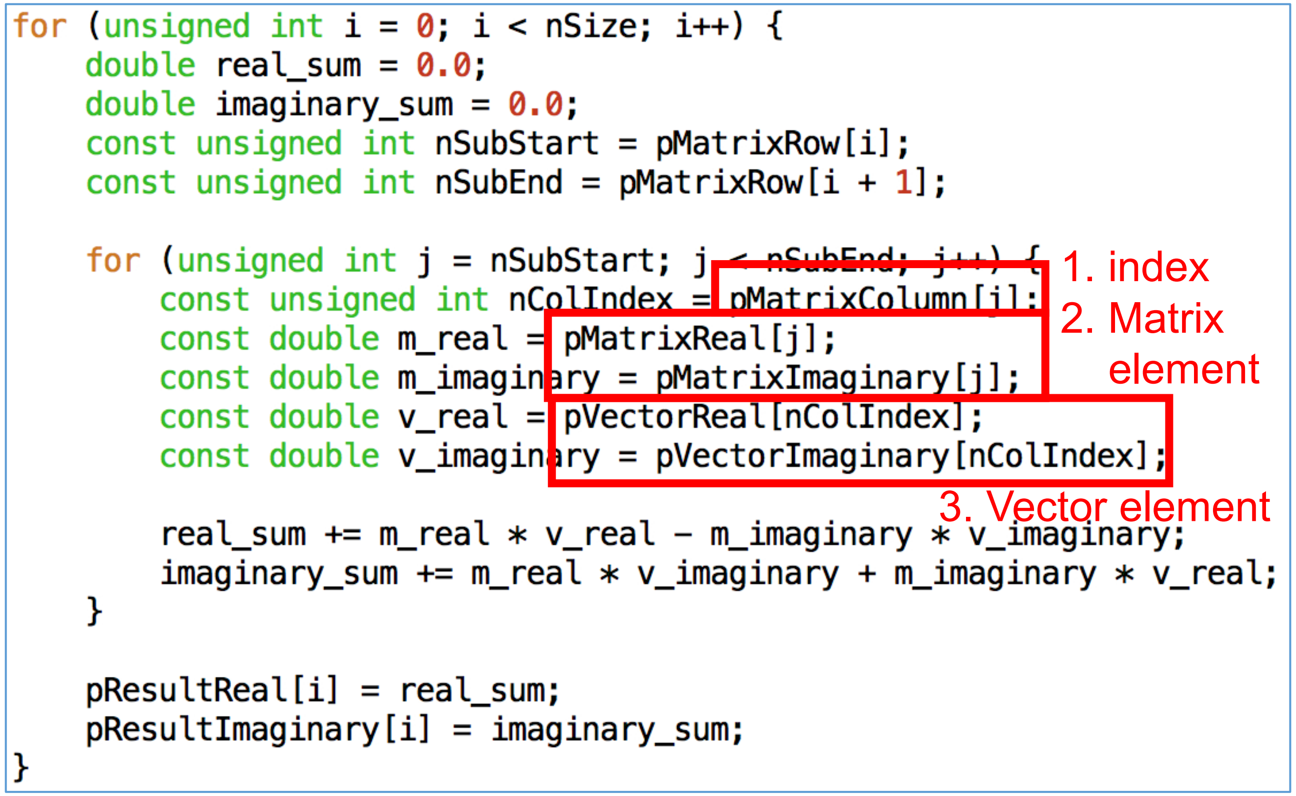


Results: **KNL consumes 2x less energy than Xeon(V4)+P100**

Performance w/ Intel® KNL Processors

When problem sizes exceed > 16GB?

Matrix-vector multiplier



```
for (unsigned int i = 0; i < nSize; i++) {
    double real_sum = 0.0;
    double imaginary_sum = 0.0;
    const unsigned int nSubStart = pMatrixRow[i];
    const unsigned int nSubEnd = pMatrixRow[i + 1];

    for (unsigned int j = nSubStart; j < nSubEnd; j++) {
        const unsigned int nColIndex = pMatrixColumn[j];
        const double m_real = pMatrixReal[j];
        const double m_imaginary = pMatrixImaginary[j];
        const double v_real = pVectorReal[nColIndex];
        const double v_imaginary = pVectorImaginary[nColIndex];

        real_sum += m_real * v_real - m_imaginary * v_imaginary;
        imaginary_sum += m_real * v_imaginary + m_imaginary * v_real;
    }

    pResultReal[i] = real_sum;
    pResultImaginary[i] = imaginary_sum;
}
```

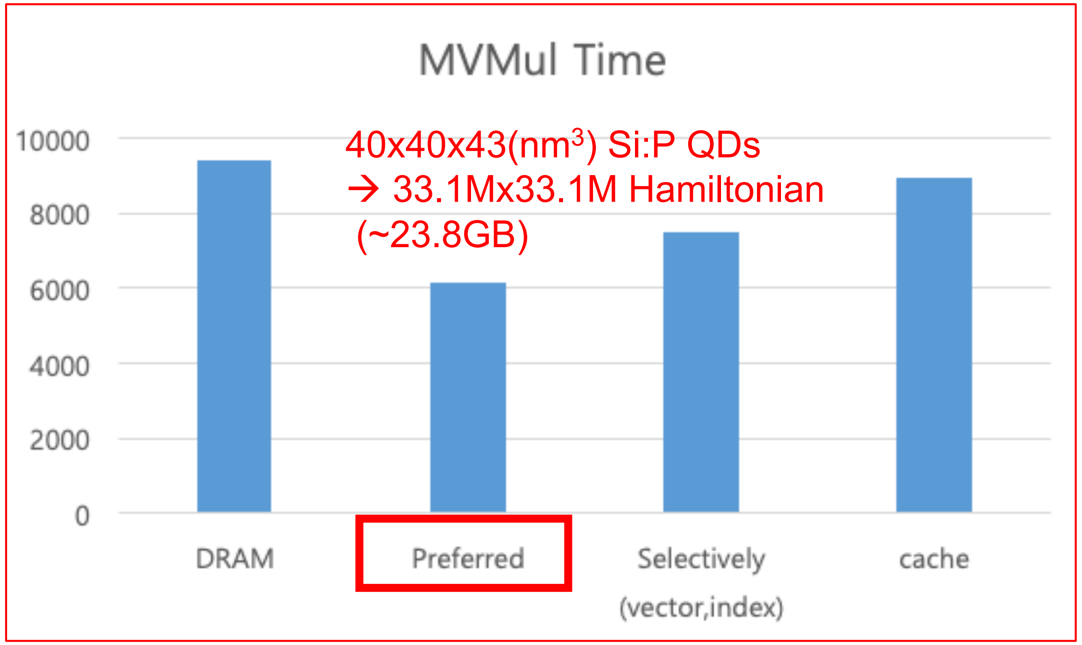
1. index
2. Matrix element
3. Vector element

Data to be saved in memory

- index: column index of matrix nonzero elements (indirect index)
- matrix and vector: matrix nonzero elements and vector elements

Available options for MCDRAM utilization

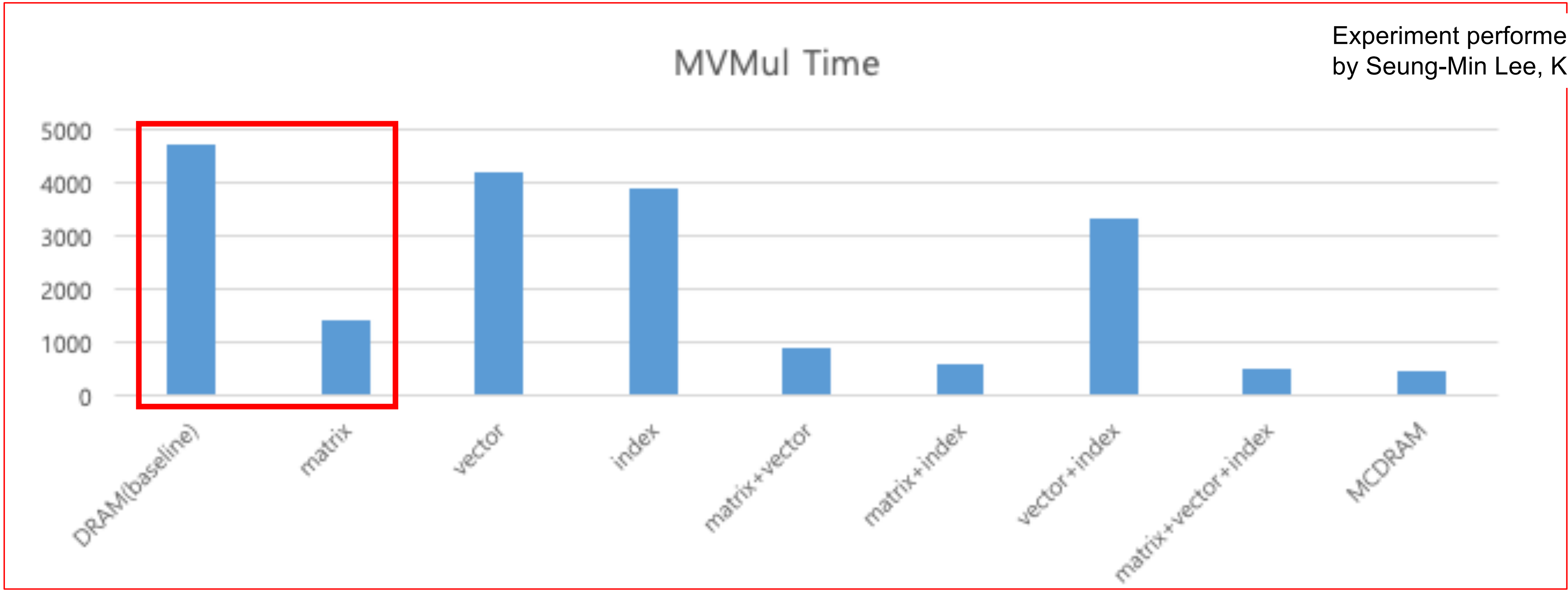
- Cache mode: use MCDRAM like L3 cache
- Preferred mode: First fill MCDRAM then go to DRAM
  → `numactl -preferred=1 …`
- Library `memkind`: use dynamic allocations in code
  → `hbw_malloc()`, `hbw_free()`



40x40x43(nm^3) Si:P QDs
→ 33.1Mx33.1M Hamiltonian
(~23.8GB)

Performance w/ Intel® KNL Processors

When problem sizes exceed > 16GB?



Experiment performed by Seung-Min Lee, KISTI

Which component would be most affected by the enhanced bandwidth of MCDRAM?

- MVMul is tested with 11GB Hamiltonian matrix

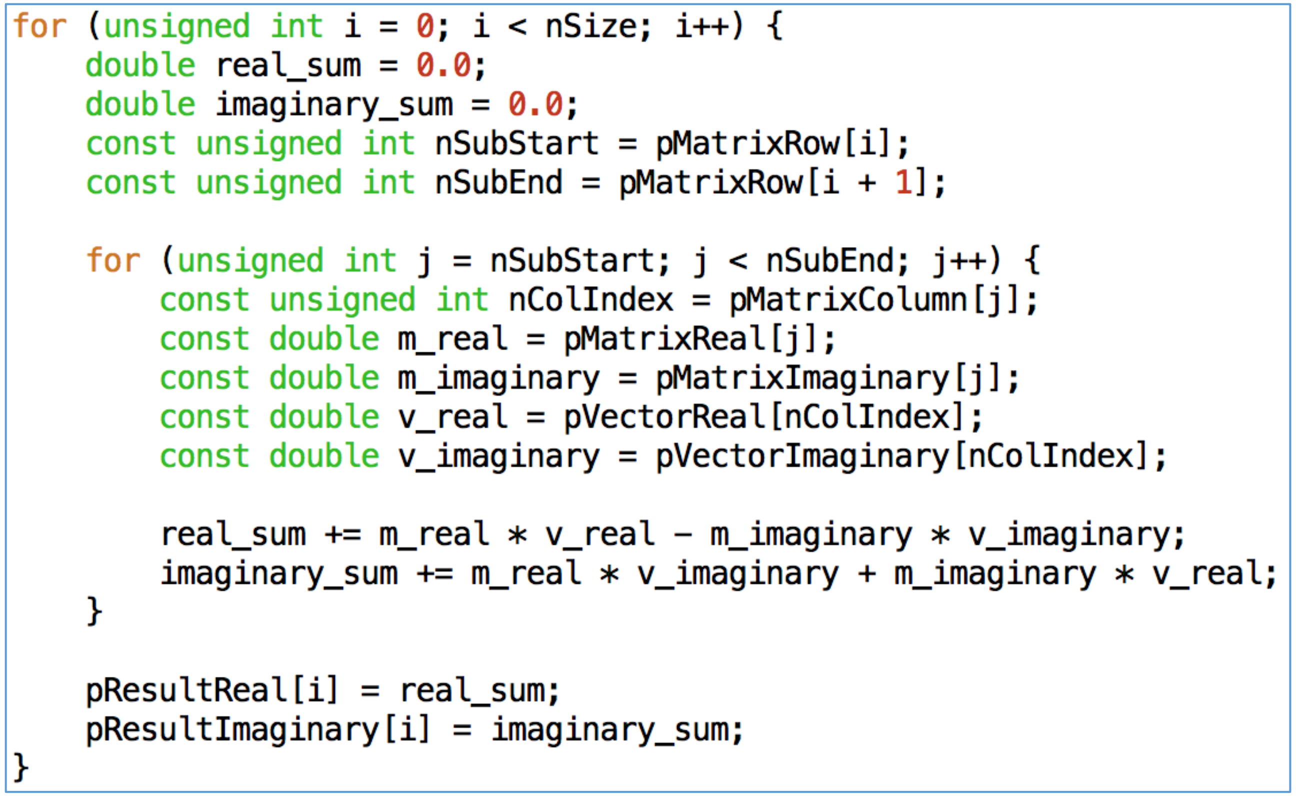
Good for us – matrix is built upon the definition of geometry!

- Matrix nonzero elements drive the most remarkable performance improvement when combined w/ MCDRAM

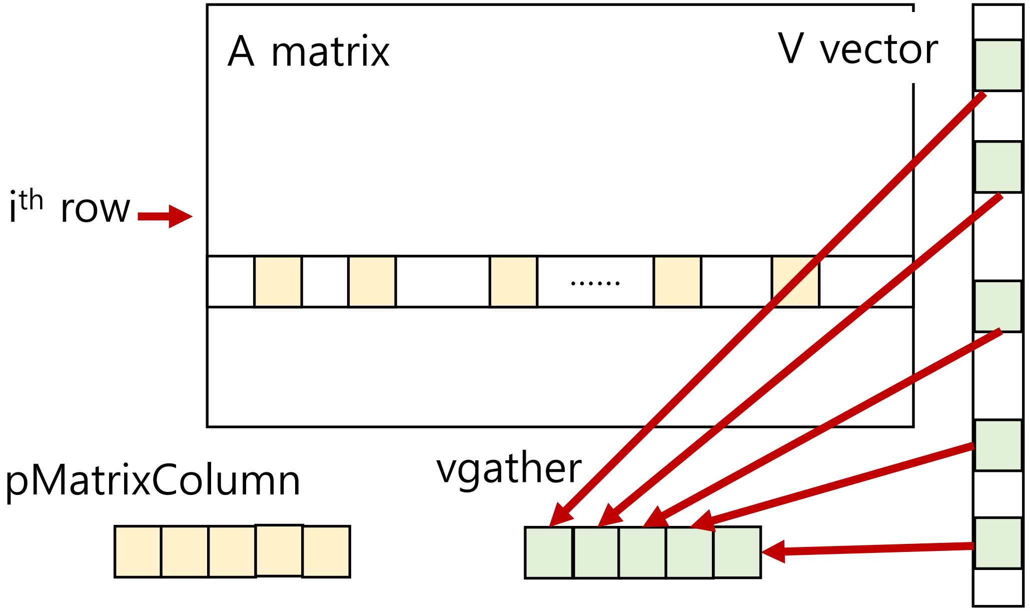
Performance w/ Intel® KNL Processors

Vectorization efficiency

Matrix-vector multiplier: Revisit



```
for (unsigned int i = 0; i < nSize; i++) {
    double real_sum = 0.0;
    double imaginary_sum = 0.0;
    const unsigned int nSubStart = pMatrixRow[i];
    const unsigned int nSubEnd = pMatrixRow[i + 1];

    for (unsigned int j = nSubStart; j < nSubEnd; j++) {
        const unsigned int nColIndex = pMatrixColumn[j];
        const double m_real = pMatrixReal[j];
        const double m_imaginary = pMatrixImaginary[j];
        const double v_real = pVectorReal[nColIndex];
        const double v_imaginary = pVectorImaginary[nColIndex];

        real_sum += m_real * v_real - m_imaginary * v_imaginary;
        imaginary_sum += m_real * v_imaginary + m_imaginary * v_real;
    }

    pResultReal[i] = real_sum;
    pResultImaginary[i] = imaginary_sum;
}
```



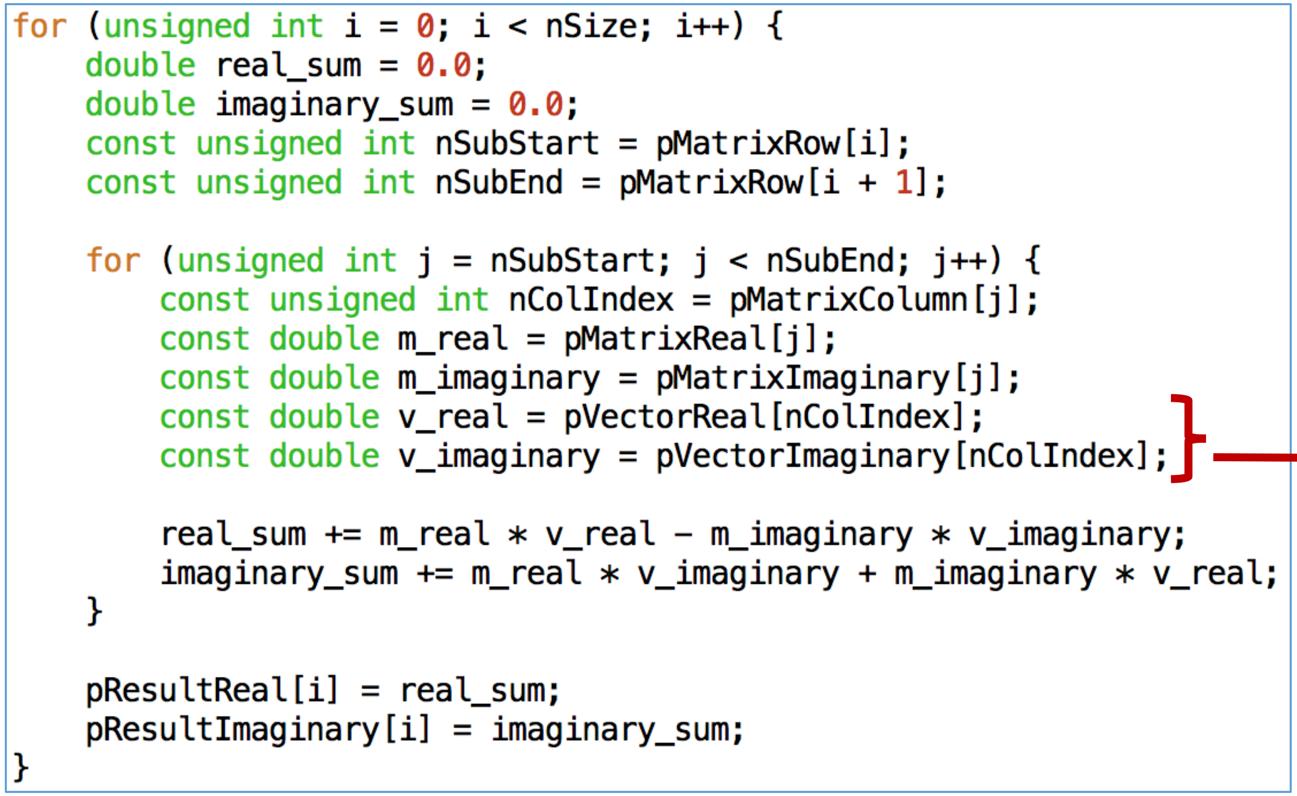
Matrix nonzeros are stored sequentially
→ But "necessary" vector elements are not, and need to be gathered in vector register first!!

- Efficiency of vectorization would not be super excellent
  → Vector elements should be "gathered" onto register before processing vectorization for matrix-vector multiplier

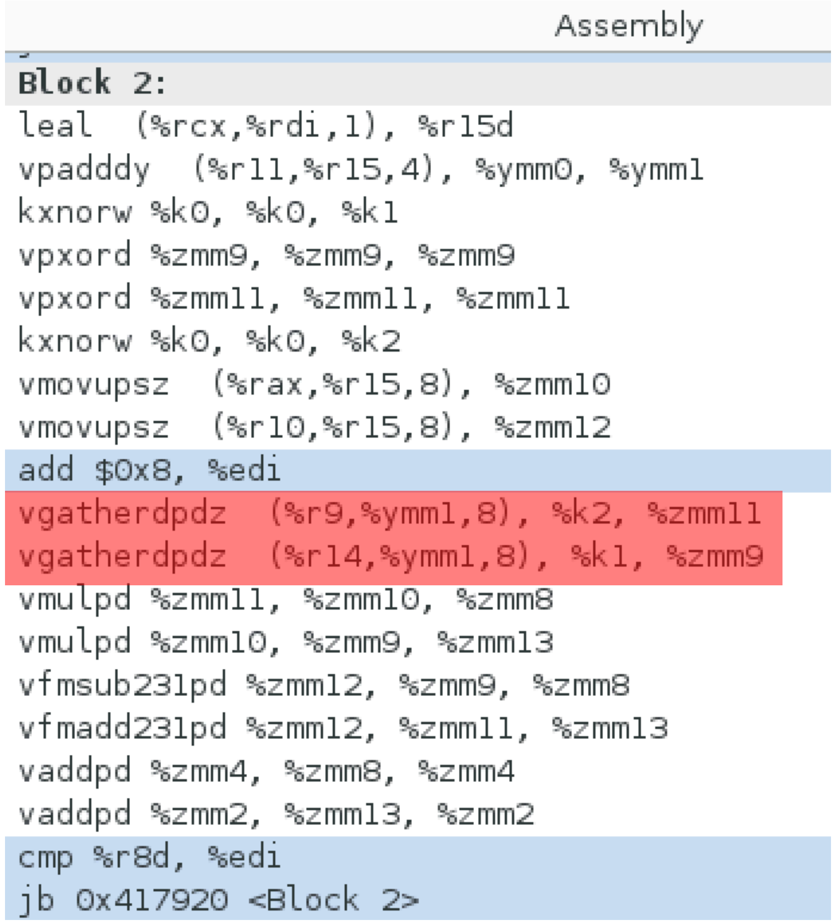
Performance w/ Intel® KNL Processors

Vectorization efficiency

Matrix-vector multiplier: Revisit



```
for (unsigned int i = 0; i < nSize; i++) {
    double real_sum = 0.0;
    double imaginary_sum = 0.0;
    const unsigned int nSubStart = pMatrixRow[i];
    const unsigned int nSubEnd = pMatrixRow[i + 1];

    for (unsigned int j = nSubStart; j < nSubEnd; j++) {
        const unsigned int nColIndex = pMatrixColumn[j];
        const double m_real = pMatrixReal[j];
        const double m_imaginary = pMatrixImaginary[j];
        const double v_real = pVectorReal[nColIndex];
        const double v_imaginary = pVectorImaginary[nColIndex];

        real_sum += m_real * v_real - m_imaginary * v_imaginary;
        imaginary_sum += m_real * v_imaginary + m_imaginary * v_real;
    }

    pResultReal[i] = real_sum;
    pResultImaginary[i] = imaginary_sum;
}
```



Assembly

```
Block 2:
leal   (%rcx,%rdi,1), %r15d
vpadddy   (%r11,%r15,4), %ymm0, %ymm1
kxnorw %k0, %k0, %k1
vpxord %zmm9, %zmm9, %zmm9
vpxord %zmm11, %zmm11, %zmm11
kxnorw %k0, %k0, %k2
vmovupsz   (%rax,%r15,8), %zmm10
vmovupsz   (%r10,%r15,8), %zmm12
add $0x8, %edi
vgatherdpdz   (%r9,%ymm1,8), %k2, %zmm11
vgatherdpdz   (%r14,%ymm1,8), %k1, %zmm9
vmulpd %zmm11, %zmm10, %zmm8
vmulpd %zmm10, %zmm9, %zmm13
vfmsub231pd %zmm12, %zmm9, %zmm8
vfmadd231pd %zmm12, %zmm11, %zmm13
vaddpd %zmm4, %zmm8, %zmm4
vaddpd %zmm2, %zmm13, %zmm2
cmp %r8d, %edi
jb 0x417920 <Block 2>
```

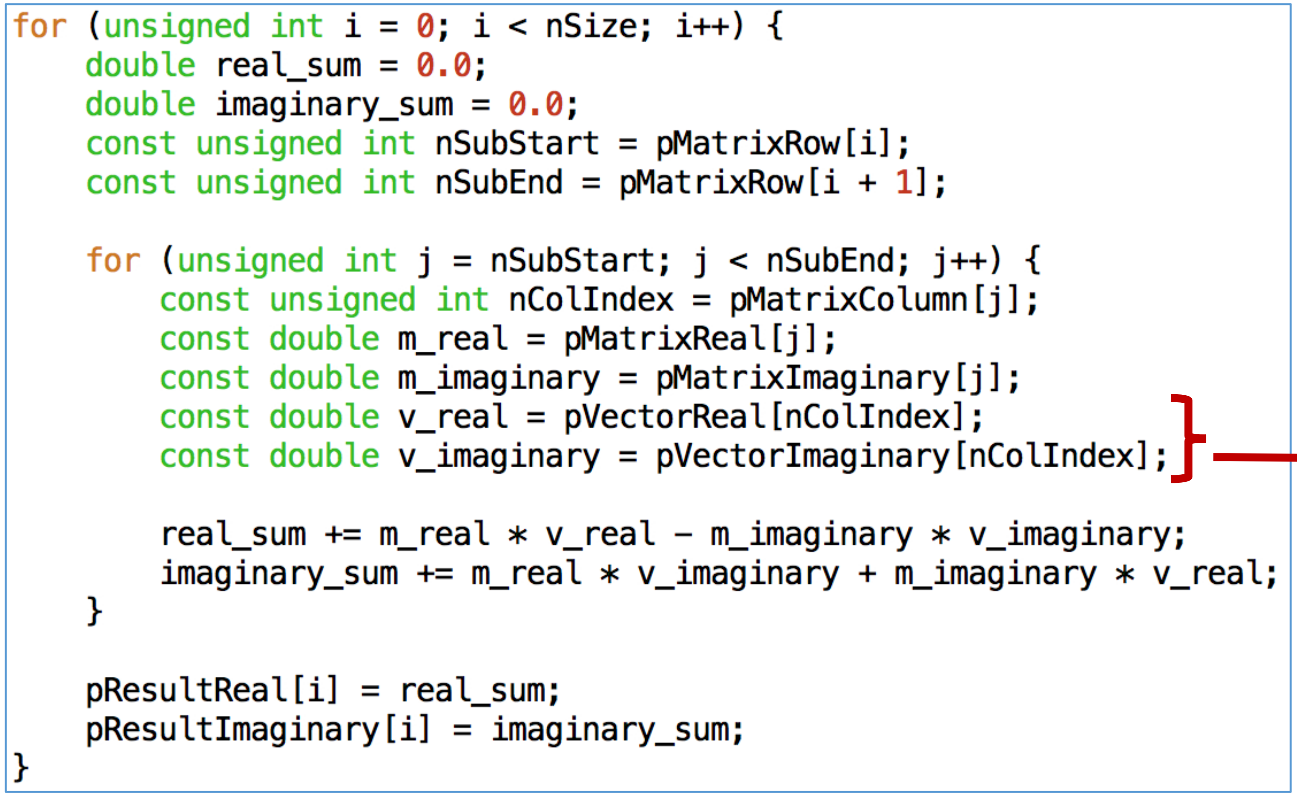
- Efficiency of vectorization would not be super excellent

  → Vector elements should be "gathered" onto register before processing vectorization for matrix-vector multiplier

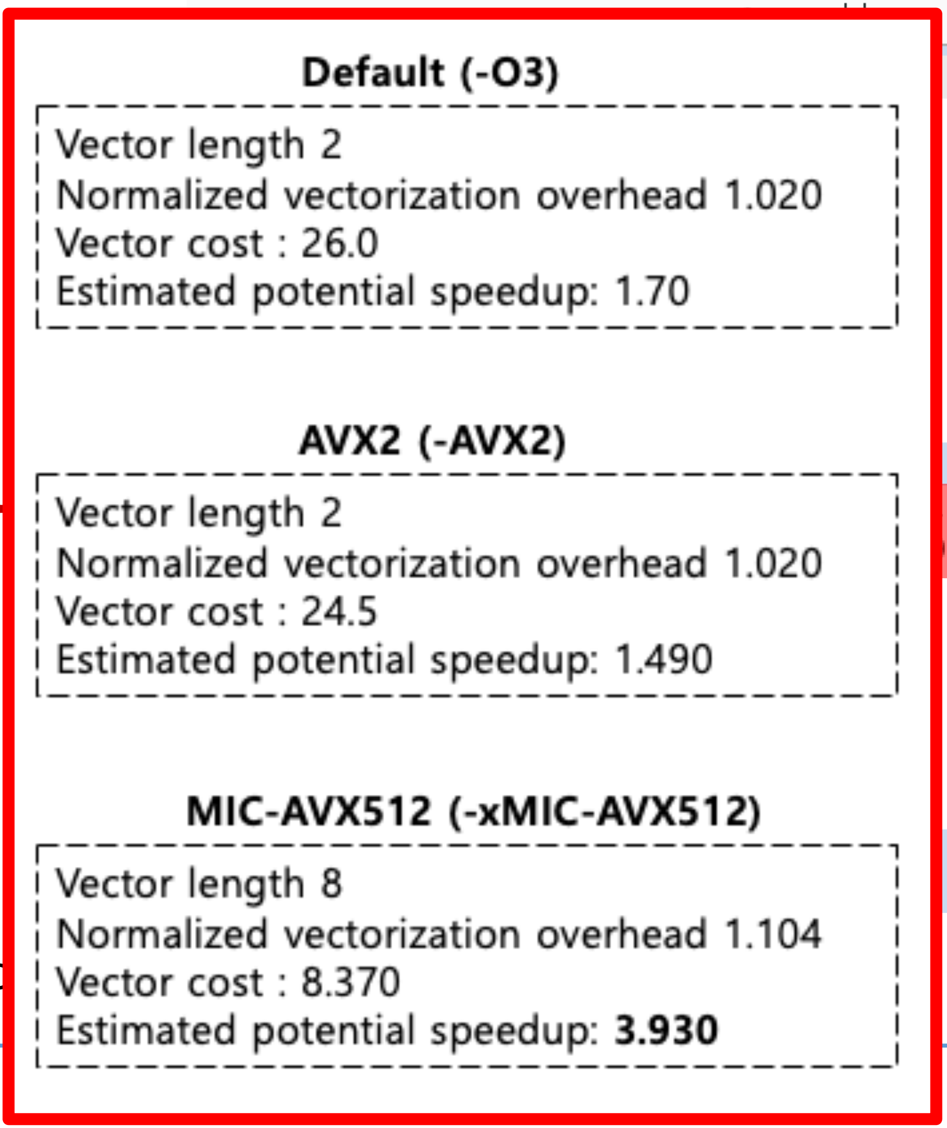
Performance w/ Intel® KNL Processors

Vectorization efficiency

Matrix-vector multiplier: Revisit



```
for (unsigned int i = 0; i < nSize; i++) {
    double real_sum = 0.0;
    double imaginary_sum = 0.0;
    const unsigned int nSubStart = pMatrixRow[i];
    const unsigned int nSubEnd = pMatrixRow[i + 1];

    for (unsigned int j = nSubStart; j < nSubEnd; j++) {
        const unsigned int nColIndex = pMatrixColumn[j];
        const double m_real = pMatrixReal[j];
        const double m_imaginary = pMatrixImaginary[j];
        const double v_real = pVectorReal[nColIndex];
        const double v_imaginary = pVectorImaginary[nColIndex];

        real_sum += m_real * v_real - m_imaginary * v_imaginary;
        imaginary_sum += m_real * v_imaginary + m_imaginary * v_real;
    }

    pResultReal[i] = real_sum;
    pResultImaginary[i] = imaginary_sum;
}
```



Default (-O3)

```
Vector length 2
Normalized vectorization overhead 1.020
Vector cost : 26.0
Estimated potential speedup: 1.70
```

AVX2 (-AVX2)

```
Vector length 2
Normalized vectorization overhead 1.020
Vector cost : 24.5
Estimated potential speedup: 1.490
```

MIC-AVX512 (-xMIC-AVX512)

```
Vector length 8
Normalized vectorization overhead 1.104
Vector cost : 8.370
Estimated potential speedup: 3.930
```

- Efficiency of vectorization would not be super excellent
  → Vector elements should be “gathered” onto register before proc ier

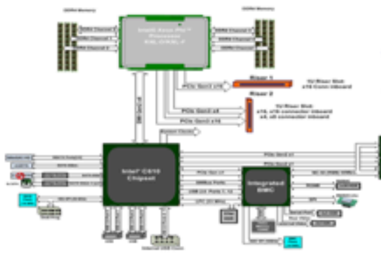
Extremely large-scale problems

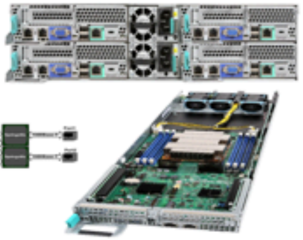
In NURION computing resource

NURION System Overview

Computing nodes Cray 3112-AA000T(2U enclosure), 8,305 KNL Computing modules

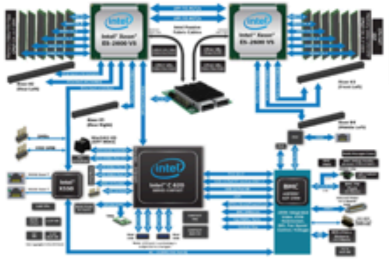
- 1x Intel Xeon Phi KNL 7250 processor
- 96GB (6x 16GB) DDR4-2400 RAM
- 1x Single-port 100Gbps OPA HFI card
- 1x On-board GigE (RJ45) port





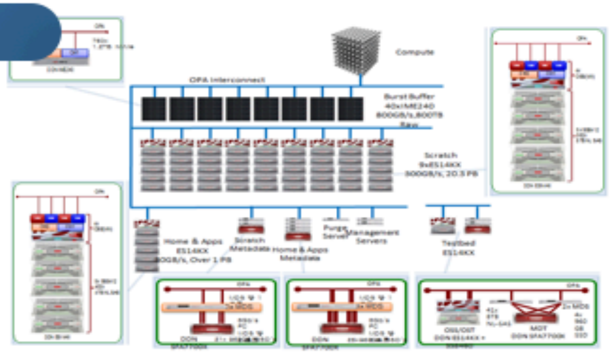
CPU-only nodes Cray 3111-BA000T(2U enclosure), 132 Skylake Computing modules

- 2x Intel Xeon SKL 6148 processors
- 192GB (12x 16GB) DDR4-2666 RAM
- 1x Single-port 100Gbps OPA HFI card
- 1x On-board GigE (RJ45) port



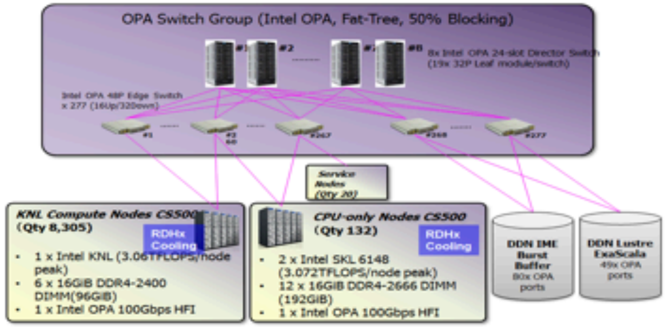
Storage 20PB SFS@300GB/s, 10PB Archiving

- Global scratch: 20PB, 0.3TB/s
  (DDN ES14KX 9ea, 360 x 8TB disk each)
- Home and application directory 1PB
- NVMe Burst Buffer: 0.8PB, 0.8TB/s
  (IME240 40ea 19 NVMe SSD each)
- Cray TSMSF and IBM TS4500



Interconnect OPA(Omni-Path Architecture), Fat-Tree, 50% Blocking

- Intel OPA High-speed interconnect switch
  274x 48-port OPA edge switches
  8x 768-port OPA core switches
- Bandwidth: 12.3 GB/sec
- Bisectional Bandwidth : 27 TB/sec
- 10^{-16} BER(Bit Error Rate), Adaptive routing


OPA Switch Group (Intel OPA, Fat-Tree, 50% Blocking)
#1
#2
#7
#8
8x Intel OPA 24-slot Director Switch (19x 32P Leaf module/switch)
Intel OPA 48P Edge Switch x 277 (16Up/32Down)
#1
#2
#267
#268
#277
Service Nodes (Qty 20)
KNL Compute Nodes CS500 (Qty 8,305)
RDHx Cooling
1 x Intel KNL (3.06TFLOPS/node peak)
6 x 16GiB DDR4-2400 DIMM(96GiB)
1 x Intel OPA 100Gbps HFI
CPU-only Nodes CS500 (Qty 132)
RDHx Cooling
2 x Intel SKL 6148 (3.072TFLOPS/node peak)
12 x 16GiB DDR4-2666 DIMM (192GiB)
1 x Intel OPA 100Gbps HFI
DDN IME Burst Buffer 80x OPA ports
DDN Lustre ExaScala 49x OPA ports

- 132 SKL (Xeon 6148) nodes / 8,305 KNL (Xeon Phi 7250) nodes
- Ranked at 13th in Top500.org as of 2018. Nov.
  → Rpeak 25.7pFLOPS, Rmax, 13.9pFLOPS. https://www.top500.org/system/179421

Extremely large-scale problems

In NURION computing resource

Description of BMT Target

- Computed lowest 3 conduction sub-bands in 2715x54x54 (nm^3) [100] Si:P square nanowire
  → contains 400 million (0.4 billion) atoms,
  → Hamiltonian matrix DOF = 4 billion x 4 billion

Computing Environment

- Intel® Xeon Phi 7250 (NURION)
  → 1.4GHz/68 cores, 96GB DRAM, 16GB MCDRAM (/node)
  → OPA (100GB)

Other Information for Code Compile and Runs

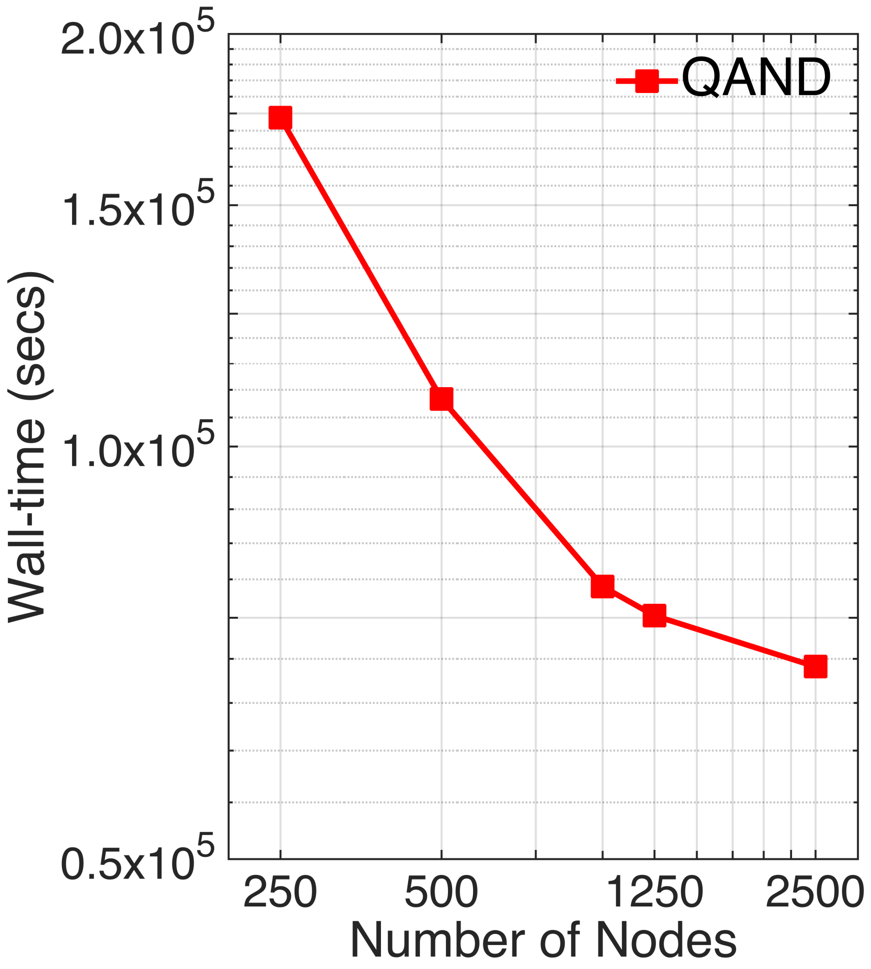
- Intel® Parallel Studio 17.0.5
- Instruction set for vectorization: MIC-AVX512
- MCDRAM allocation: numactl –preferred=1
- OPA fabric. 4 MPI processes / 17 threads per node
- Memory Placement Policy Control
  → `export I_MPI_HBW_POLICY = hbw_bind,hbw_preferred,hbw_bind`
  (HBW memory for RMA operations and for Intel® MPI Library first. If HBW memory is not available, use local DDR)
  RMA: Remote Memory Access (for MPI communications)

```
module purge
module load craype-network-opa
module load intel/17.0.5
module load impi/17.0.5

export NUMACTL="numactl --preferred=1"
export I_MPI_HBW_POLICY=hbw_bind,hbw_preferred,hbw_bind
export I_MPI_FABRICS=ofi

ulimit -s unlimited
cd $PBS_O_WORKDIR
cat $PBS_NODEFILE
time mpirun $NUMACTL ...
```

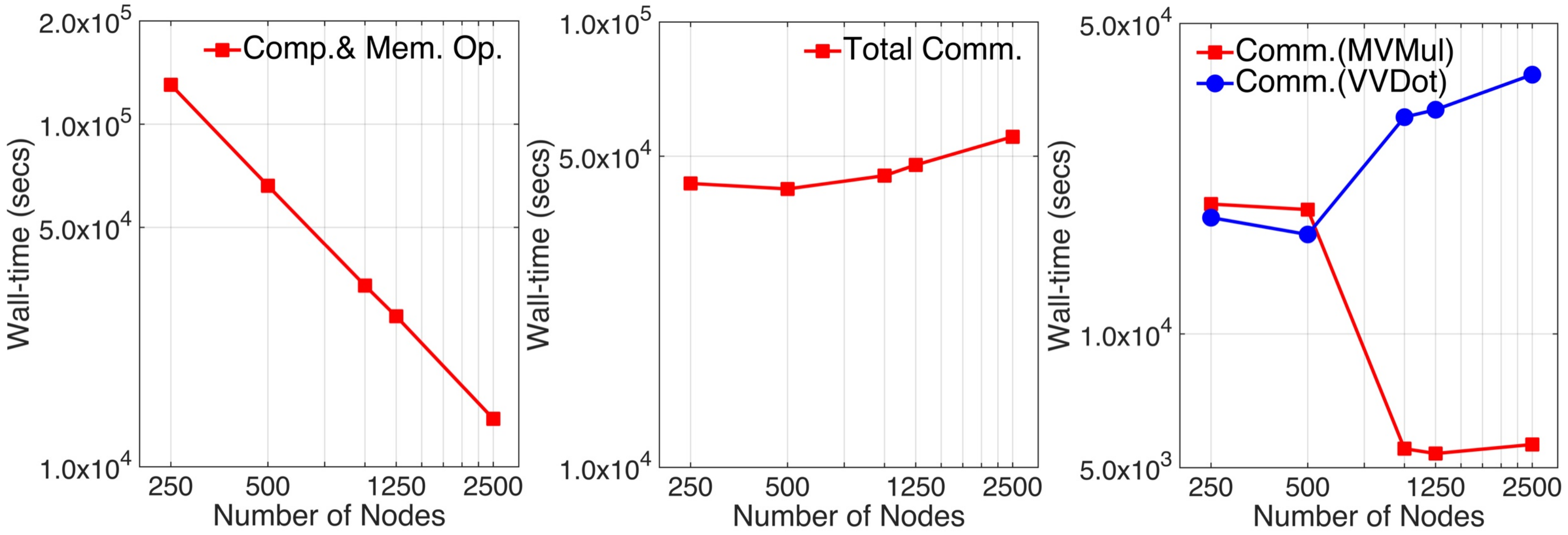
Snapshot of a PBS script

Extremely large-scale problems

In NURION computing resource



To be presented in IEEE CLUSTER (2019)



Remarks

- Scalability up to 2500 KNL nodes (~30% of the entire KNL resources in NURION)
  → Computations (including MVMul and VVDot) and Memory Operations.
  → Communications (MVMul and VVDot Communications)
- Collective communication (Allreduce(MPI_SUM)) serves as a bottleneck when computing nodes > 500 are involved.

Summary

KISTI Intel® Parallel Computing Center

- Introduction to Code Functionality
- Main Numerical Problems and Strategy of Development
- Performance (speed and energy consumption) in a single KNL node

  → Benefits against the case of CPU + 2xP100 GPU devices
- Performance in extremely huge computing environment

  → Strong scalability up to 2,500 KNL nodes in NURION system
- (Appendix) Strategy of Performance Improvement towards PCI-E devices
- (Appendix) Applications: Perovskite Optoelectronics

Thanks for your attention!!

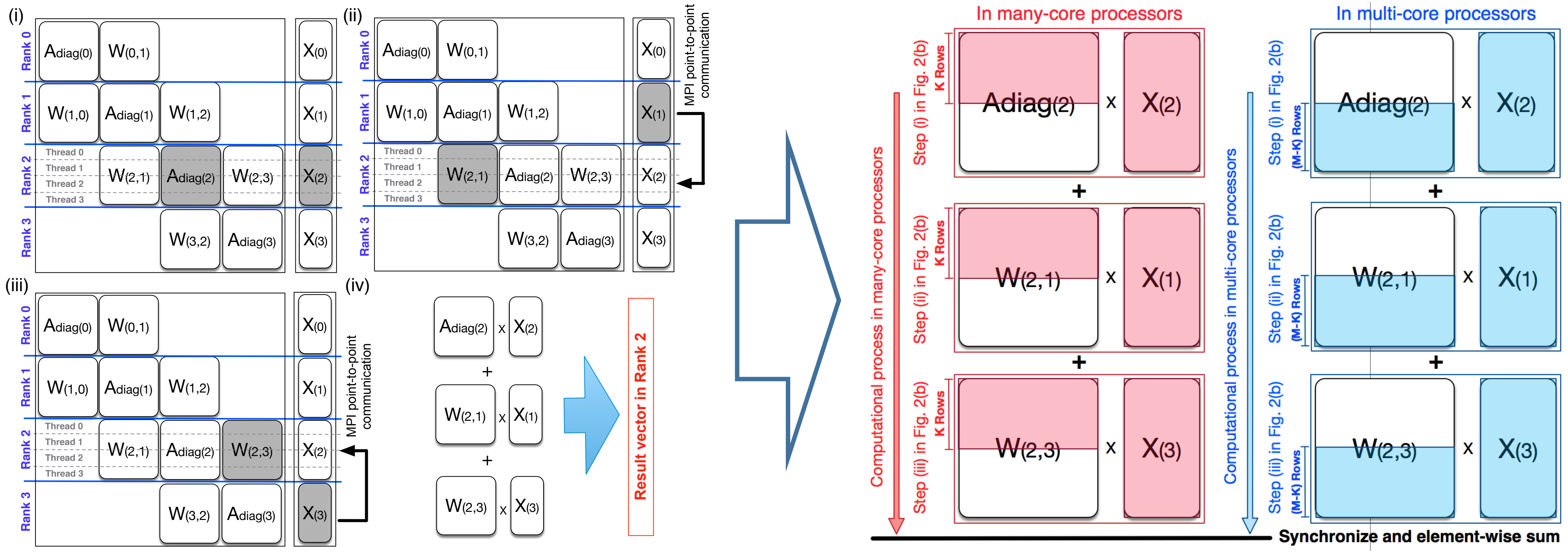
Strategy for offload-computing

Asynchronous Offload (for Xeon(V4) + KNC, Xeon(V4) + GPU)

The real bottleneck of computing: Overcome with asynchronous offload

H. Ryu et al., Comp. Phys. Commun. (2016) (http://dx.doi.org/10.1016/j.cpc.2016.08.015)

- Vector dot-product is not expensive: All-reduce, but small communication loads
- Vector communication is not a big deal: only communicates between adjacent layers
- Sparse-matrix-vector multiplication is a big deal: Host and PCI-E device shares computing load



Strategy for offload-computing

Data-transfer and Kernel Functions for GPU Computing

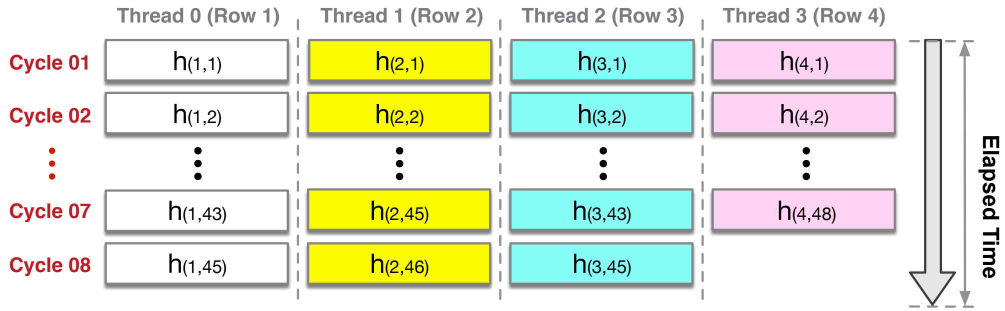
Data-transfer between host and GPU Devices

- 3x increased bandwidth with **pinned memory**
- Overlap of computation and data-transfer with **asynchronous streams**

Speed-up of GPU Kernel Function (MVMul)

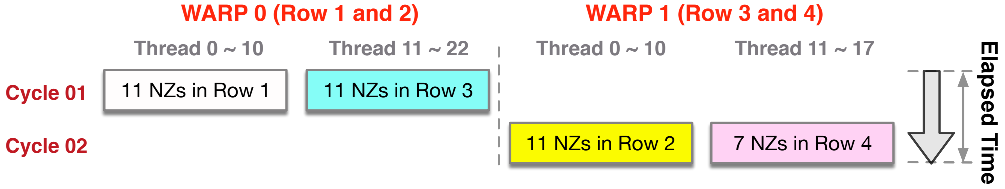
- Treating several rows at one time with WARPs

Data-Access with a thread-base (no WARPs)



	Thread 0 (Row 1)	Thread 1 (Row 2)	Thread 2 (Row 3)	Thread 3 (Row 4)
Cycle 01	$h_{(1,1)}$	$h_{(2,1)}$	$h_{(3,1)}$	$h_{(4,1)}$
Cycle 02	$h_{(1,2)}$	$h_{(2,2)}$	$h_{(3,2)}$	$h_{(4,2)}$
⋮	⋮	⋮	⋮	⋮
Cycle 07	$h_{(1,43)}$	$h_{(2,45)}$	$h_{(3,43)}$	$h_{(4,48)}$
Cycle 08	$h_{(1,45)}$	$h_{(2,46)}$	$h_{(3,45)}$	

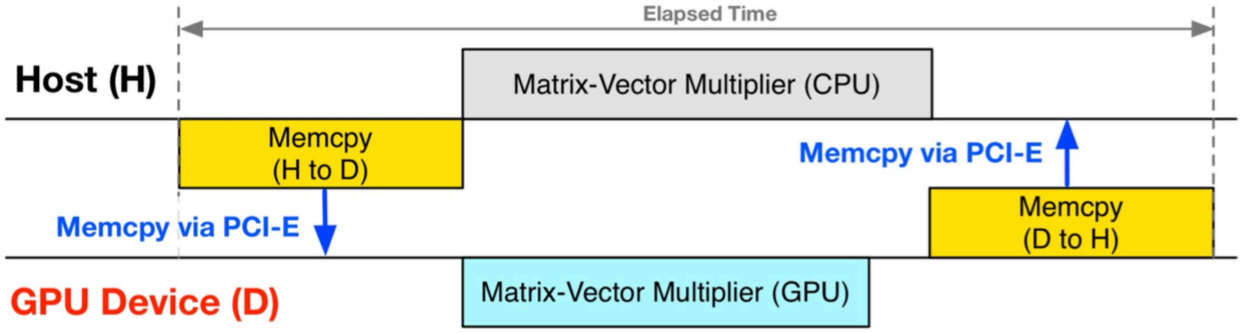
Data-Access with a WARP-base



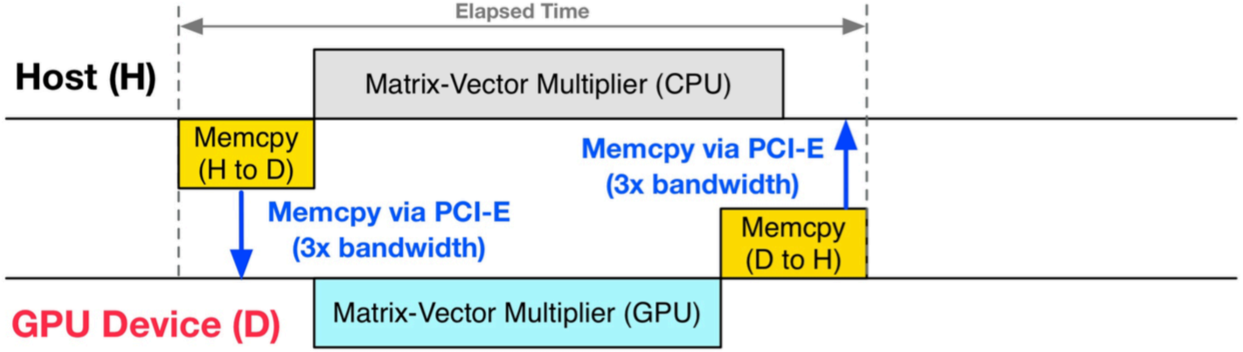
	WARP 0 (Row 1 and 2)		WARP 1 (Row 3 and 4)	
	Thread 0 ~ 10	Thread 11 ~ 22	Thread 0 ~ 10	Thread 11 ~ 17
Cycle 01	11 NZs in Row 1	11 NZs in Row 3		
Cycle 02			11 NZs in Row 2	7 NZs in Row 4

H. Ryu et al., J. Comp. Elec. (2018)
(http://dx.doi.org/10.1007/s10825-018-1138-4)

[Synchronous Data Transfer with Pageable Memory]



[Asynchronous Data Transfer with Pinned Memory]



Applications: MHP Optoelectronics

Metal Halide Perovskites (MHPs)

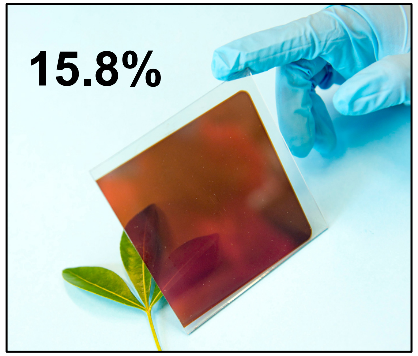
- **<u>Any materials</u>** with the same type of crystal structure as calcium titanium oxide ($CaTiO_3$)

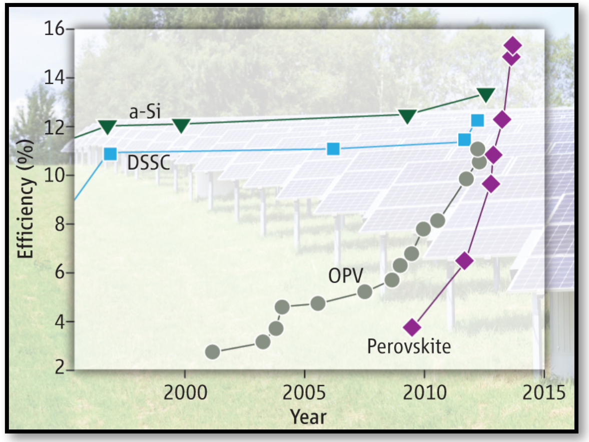
  → **ABX_3** structure (general chemical formula)

- **$CH_3NH_3PbX_3$** (Methylammonium Lead Halides)

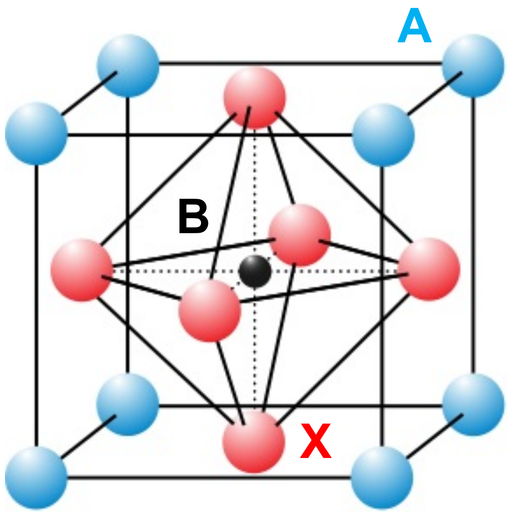
  → Highly efficient photovoltaic devices (X=Iodide)))

  → Cost-efficient / **<u>Bandgap-tunability</u>** for visible lights





Science **342**, 317-318



- **$CsPbX_3$** (Cesium Lead Halides)

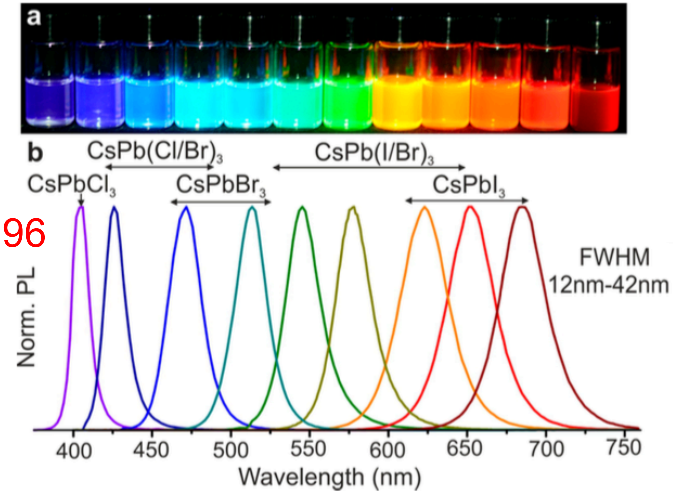
  → All-inorganic material: enhanced stability

  → **<u>Bandgap-tunability</u>** is as good as $MAPbX_3$

- **Halide-controlled bandgap**

  → Iodide (I), Bromide (Br), and Chloride (Cl)

  → Attractive for Light-emitting diode designs

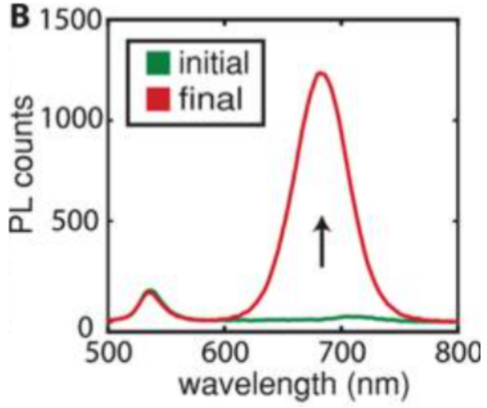


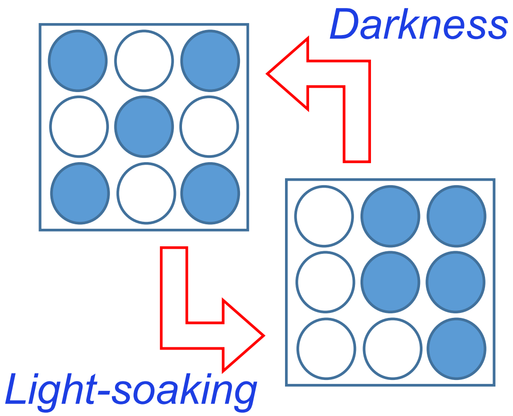
Nano Lett. **15**, 3692-3696

Issue: Phase Separation in Mixed Halides

A problem in getting lights of stable emission wavelengths

Photoirradiation



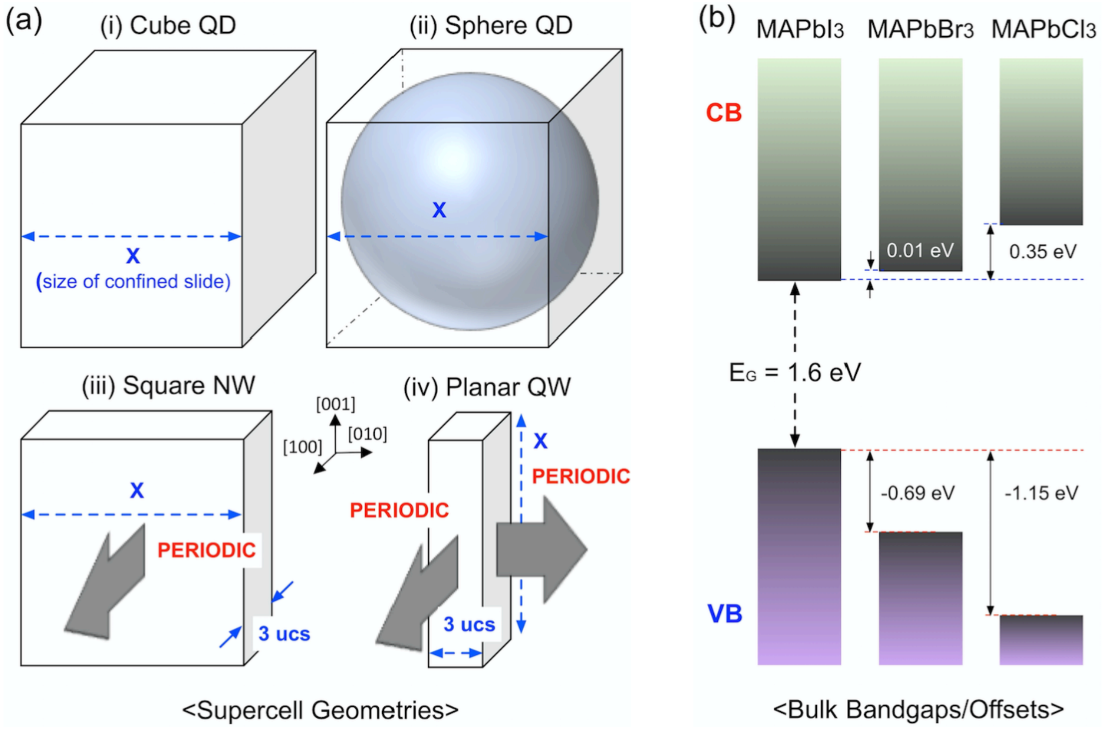


Nano Lett. **17,** 1028-1033

Questions to be answered with TB simulations

- Why does a phase separation drive a red-shift?
- How to reduce the effects of a photoirradiation (red-shift)?
  → In viewpoints of structural / compositional engineering

pm3m



H. Ryu et al., J. Phys. Chem. Lett. 10, 2745-2752 (2019)

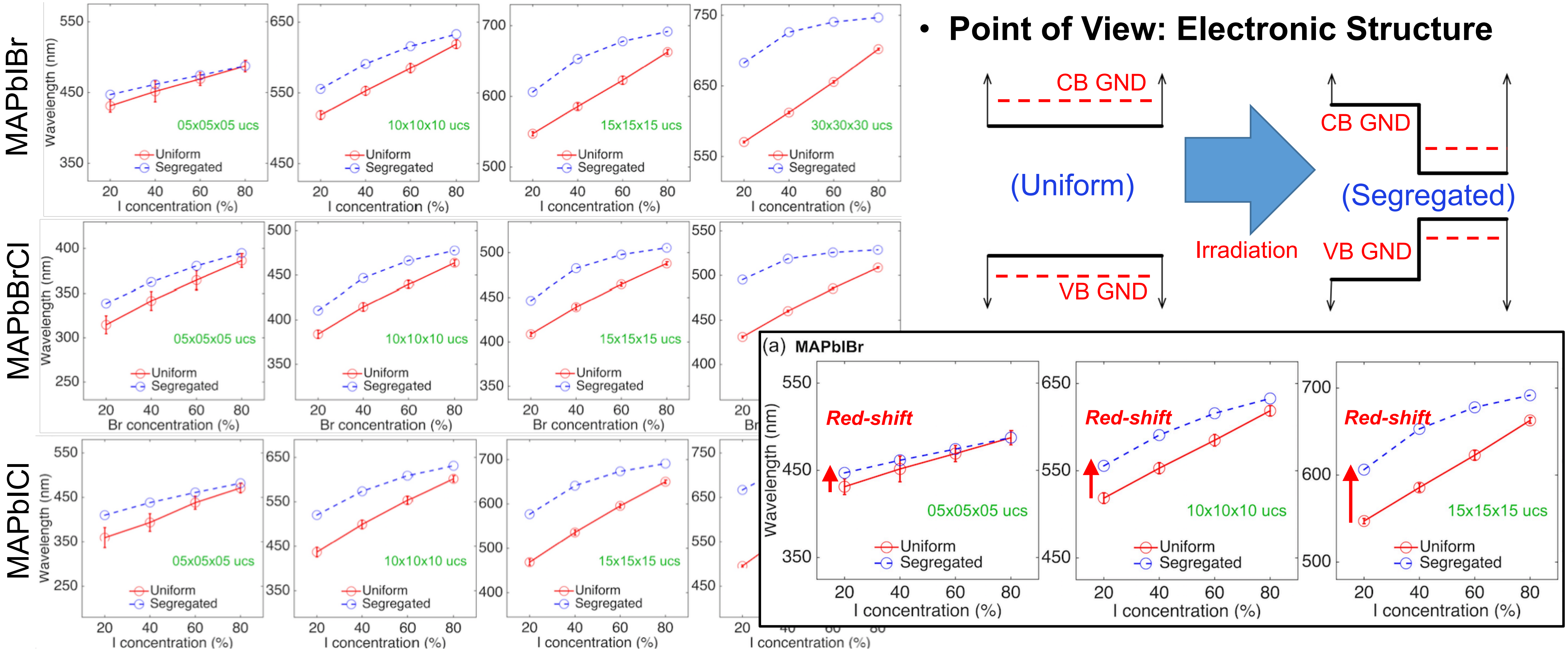
Approach of Simulations

- A 8-band (sp^3 w/ S.O.) tight-binding model
- Parameters are fitted to reproduce DFT-known bulk band gap energies w/ offsets
- 4 types of supercell geometries
  → Cube and Sphere Quantum Dots (QDs)
  → Square Nanowires (NWs)
  → Nano-platelets (Quantum Wells ,QWs)

Band Gap Details

MHPs with binary halide mixtures

- **Point of View: Electronic Structure**


MAPbIBr
Wavelength (nm)
I concentration (%)
05x05x05 ucs
10x10x10 ucs
15x15x15 ucs
30x30x30 ucs
Uniform
Segregated
MAPbBrCl
Wavelength (nm)
Br concentration (%)
05x05x05 ucs
10x10x10 ucs
15x15x15 ucs
Uniform
Segregated
MAPbICl
Wavelength (nm)
I concentration (%)
05x05x05 ucs
10x10x10 ucs
15x15x15 ucs
Uniform
Segregated
CB GND
(Uniform)
VB GND
Irradiation
CB GND
(Segregated)
VB GND
(a) MAPbIBr
Wavelength (nm)
Red-shift
Red-shift
Red-shift
05x05x05 ucs
10x10x10 ucs
15x15x15 ucs
Uniform
Segregated
I concentration (%)