An optimization of search for neighbour-particle in MPS method for Xeon, Xeon Phi and GPU by using directives

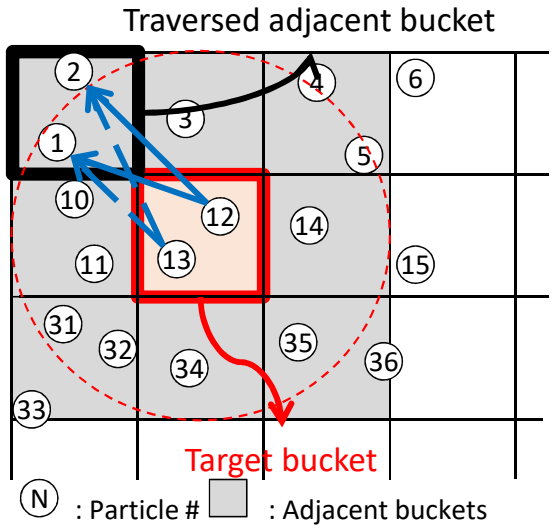
Takaaki Miyajima †, Kenichi Kubota ‡ and Naoyuki Fujita ‡

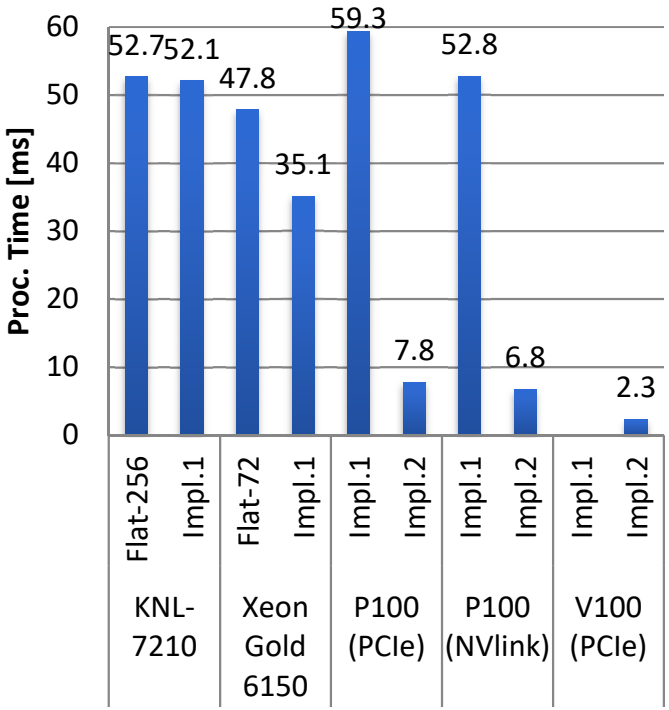
† RIKEN R-CCS,

‡ Japan Aerospace Exploration Agency

Short Summary

- ***P-Flow* : JAXA's in-house MPS program**
  - ✓ Apply MPS method to aerospace field
  - ✓ Neighbour-particle search is a bottle neck
- **Propose two optimizations while considering data size for 1 bucket**
  - ✓ 1 bucket requires 17.9 KB at maximum
  - ✓ Opt. 1: 1 thread/ 1 bucket
  - ✓ Opt. 2: 1 thread block / 1 bucket
- **Evaluate on Xeon / Xeon Phi / P100**
  - ✓ Intel Xeon Skylake-SP : 104.1[ms]
  - ✓ Intel Xeon Phi KNL : 35.1[ms]
  - ✓ NVIDIA Tesla P100(NVlink) : 6.8[ms]
  - ✓ NVIDIA Tesla P100(PCIe) : 7.8[ms]
  - ✓ NVIDIA Tesla V100(PCIe) : 2.3[ms]





Outline

1. ***P-Flow*: JAXA's in-house MPS program**
   i. Moving Particle Simulation (MPS) method
   ii. P-Flow and its internal structure
2. **Proposed optimization techniques**
   i. Calculates required data size for 1 bucket
   ii. Opt. 1: 1 thread/ 1 bucket
   iii. Opt. 2: 1 thread block / 1 bucket
3. **Implementation and evaluation**
   i. Xeon/Xeon Phi
   ii. Tesla P100
   iii. Xeon/Xeon Phi, Tesla P100
4. **Discussion and summary**

Motivation: Unsteady-state simulation

- **Brand new method is required to simulate unsteady-state (taking off, landing, or circling) of airplane**
  - ✓ Goal: Simulate interaction between raindrops and aircraft for water ingestion test. Raindrops give a negative effect on wings and tires.

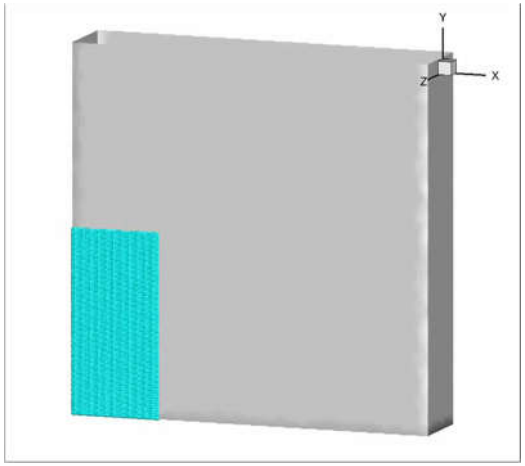
Water ingestion test is mandatory for new airplane
Movie: https://www.airbus.com

Wet runway water ingestion test of Honda jet.
Photo: https://hondanews.com/hondajet/photos

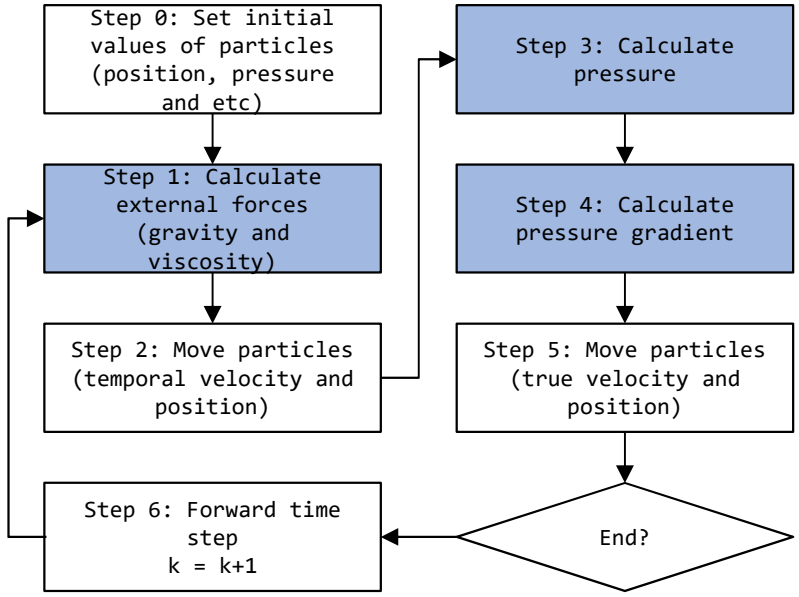
We adopt Moving Particle Semi-implicit (MPS) method for simulating raindrops

Moving Particle Simulation (MPS) Method

- **Overview**
  - ✓ MPS becomes popular in CFD area
    - ➢ Developed for incompressible flow.
  - ✓ Particle-base simulation
    - ➢ Not a stencil computation
  - ✓ Target fluids are divided to thousands of particles
    - ➢ Each particle interacts with its neighbour-particle.
  - ✓ The neighbour-particle search part is a main bottleneck
    - ➢ MPS compute neighbour-particle search five times in each time integration



Benchmark simulation by *P-Flow*



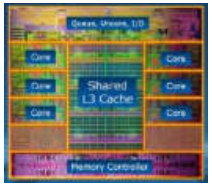
Processing step of MPS

P-Flow: JAXA's in-house MPS program

- **Features w.r.t computer science**
  - ✓ 5,000 lines of Fortran90, single-prec. FP is used
  - ✓ Data structure: SoA for physical quantities, AoS for bucket management
  - ✓ Memory-bound (indirect mem. access) application

Japan Aerospace Exploration Agency

- **Requirement**
  - ✓ Computational scientist implements and tests new method
  - ✓ Need to evaluate Xeon, Xeon Phi, and GPU
  - ✓ Data structure should be kept as it is
    - ➢ Deep copy
  - ✓ MPI+X: MPI+OpenMP/OpenACC
    - ➢ CUDA-aware MPI (!$acc host_data use_device())



Intel "Xeon"

Intel "Xeon Phi"

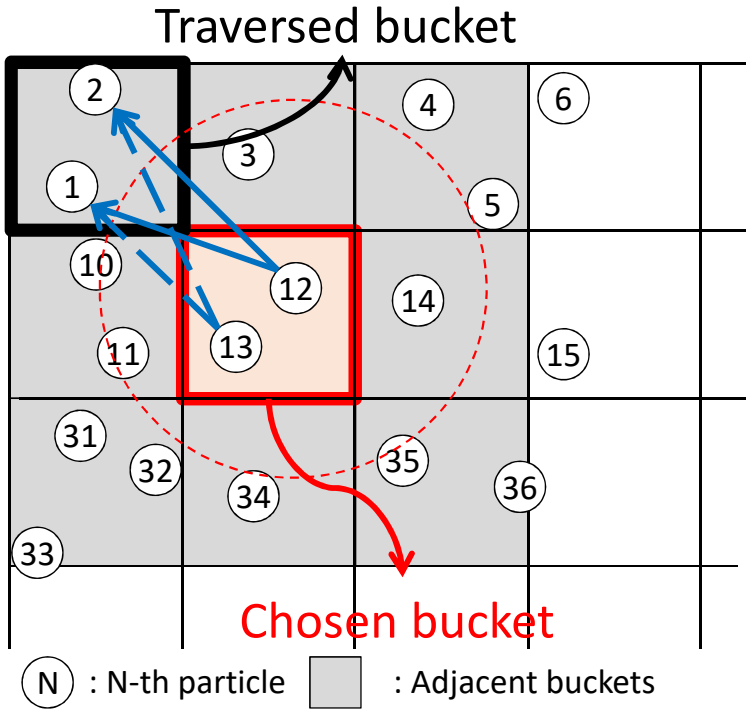
NVIDIA "Tesla GPU"

Balance performance and simplicity :)

Neighbour-particle search

【Buckets】

- Decompose simulation space into squares called “bucket”
- The volume of bucket is equal to 3*3*3 particles
- Effect radius (cut-off distance) is 3 particles



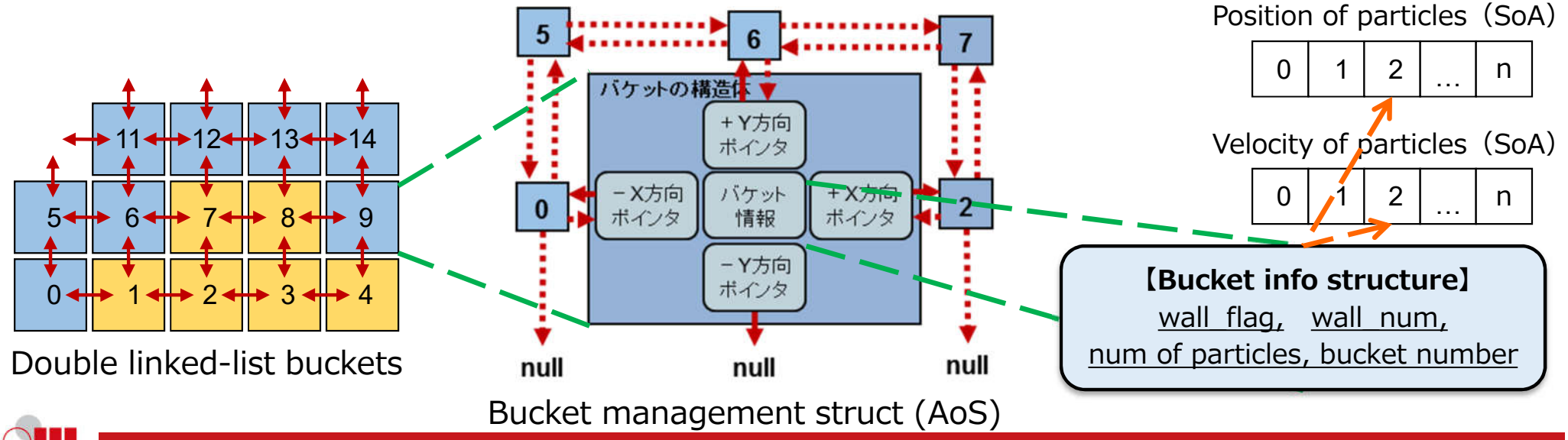
【Processing step】

1. Traverse buckets
2. Pickup a target particle (red)
3. Traverse adjacent 3^3 buckets
   - ✓ No fiexed order to traverse bucket
4. Search particles in a bucket
5. Calculate distance and weight between the target particle
6. Accumulate weighted physical value to a target particle
   - ✓ No fixed order to accumulate physical value

Maximum number of particles in a bucket is 33

Linked-list for bucket management

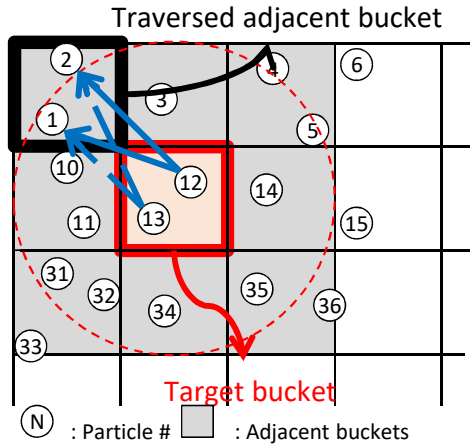
- **Adjacent buckets are expressed as doubly linked-lists**
  - ✓ Iterator (not for-loop) is used to traverse adjacent buckets
  - ✓ AoS for bucket management, SoA for physical quantities
  - ✓ Deep copy is required !!
- **Good for domain decomposition with multiple nodes**
  - ✓ Simple indexing can not deal with domain decomposition
  - ✓ Each bucket doesn't need to store all the adjacency information
  - ✓ Need to refer bucket management structure to access particles



Implementation of neighbour-particle search in *P-Flow*

```
! Loop-1: 対象バケットを選択 ----------------------
do ib_lc = 1, bkts%num
   call get_bkt_info(...)       ! バケット情報と
   call get_bkt_ijk(...)        ! 各方向の番号を取得
   ! Loop-2: 選択したバケット内の粒子について ------
   do n = 1, np
      l          = pc_id_lc + n ! 粒子のインデックス
      m          = Lcr(l)       ! 更新粒子のID取得
      nden_temp = 0             ! 物理量初期化
      call it_init(...)         ! イテレータ初期化
      ! Loop-3: 隣接3x3のバケット探索 -------------
      do while(it_next(...))
         call get_bkt_info(...)! バケット情報取得
         ! Loop-4: 探索バケット内の粒子について ---
         do nn = 1, nnp
            ll = pc_id_lc2 + nn
            if(irank == myrank) then
               mm = Lcr(ll)     ! bktsのindexを使う
               if(mm == m) cycle
            else
               mm = Lnr(ll)     ! halsのindexを使う
            end if
            ! 対象粒子の距離と重みを算出
            r = get_distance(x(:, mm), x(:, m))
            dn         = wgt_func(r, re)
            ! 物理量を加算
            nden_temp = nden_temp + dn
         end do
      end do
      nden(m) = nden(m) + nden_temp
   end do
end do
```

- **Loop-1: Choose a target bucket**
- **Loop-2: Pickup a target particle (red) in the bucket**
- **Loop-3: Traverse 3*3*3 adjacent buckets (in no particular order)**
- **Loop-4: Search particles in a bucket**
  - ✓ 4-1. Calculate distance and weight between the target particle
  - ✓ 4-2. Accumulate weighted physical value to a target particle (in no particular order)

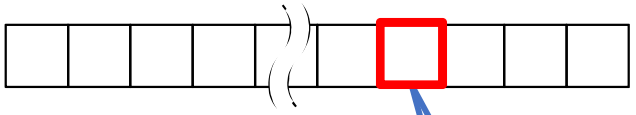


It's four nested loop.
Which loop(s) should be assigned to thread or thread block?

Memory access pattern

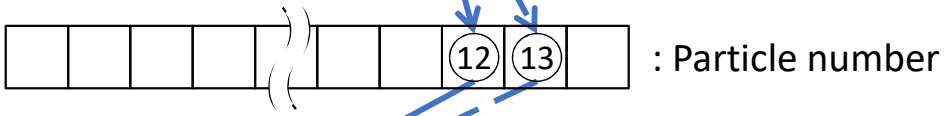
```
! Loop-1: 対象バケットを選択 ----------------------
do ib_lc = 1, bkts%num
   call get_bkt_info(...)       ! バケット情報と
   call get_bkt_ijk(...)        ! 各方向の番号を取得
   ! Loop-2: 選択したバケット内の粒子について ------
   do n = 1, np
      l          = pc_id_lc + n ! 粒子のインデックス
      m          = Lcr(l)       ! 更新粒子のID取得
      nden_temp = 0             ! 物理量初期化
      call it_init(...)         ! イテレータ初期化
      ! Loop-3: 隣接3x3のバケット探索 ------------
      do while(it_next(...))
         call get_bkt_info(...)! バケット情報取得
         ! Loop-4: 探索バケット内の粒子について ---
         do nn = 1, nnp
            ll = pc_id_lc2 + nn
            if(irank == myrank) then
               mm = Lcr(ll)     ! bktsのindexを使う
               if(mm == m) cycle
            else
               mm = Lnr(ll)     ! halsのindexを使う
            end if
            ! 対象粒子の距離と重みを算出
            r = get_distance(x(:, mm), x(:, m))
            dn          = wgt_func(r, re)
            ! 物理量を加算
            nden_temp = nden_temp + dn
         end do
      end do
      nden(m) = nden(m) + nden_temp
   end do
end do
```

- Loop-1: Choose a target bucket



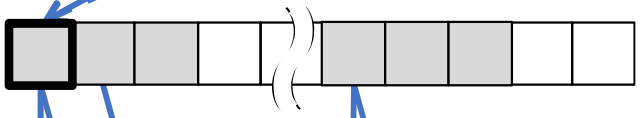
In-direct access

- Loop-2: Pickup a target particle (red) in the bucket

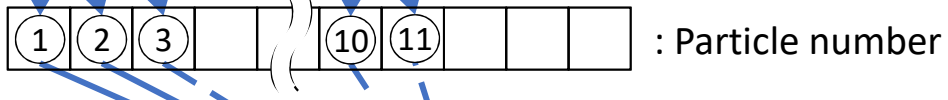


Indefinite loop

- Loop-3: Traverse 27 adjacent buckets

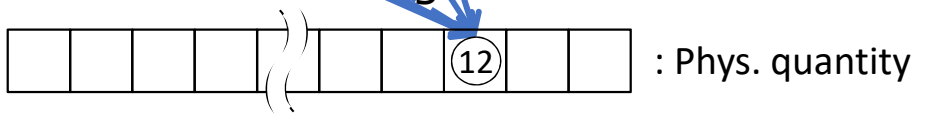


- Loop-4: Search particles in a bucket



Accumulation

- 4-1 and 2: Weight calc. and accum.

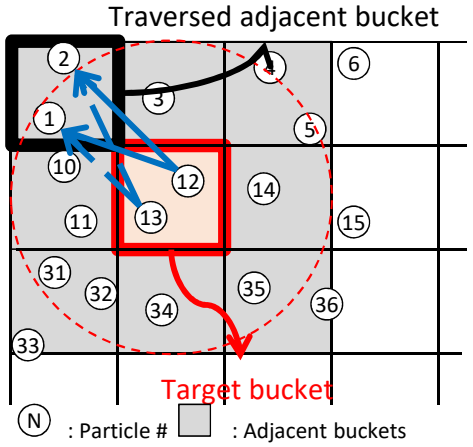


Outline

1. ***P-Flow*: JAXA's in-house MPS program**
   i. Moving Particle Simulation (MPS) method
   ii. P-Flow and its internal structure
2. **Proposed optimization techniques**
   i. Calculates required data size for 1 bucket
   ii. Opt. 1: 1 thread/ 1 bucket
   iii. Opt. 2: 1 thread block / 1 bucket
3. **Implementation and evaluation**
   i. Xeon/Xeon Phi
   ii. Tesla P100
   iii. Xeon/Xeon Phi, Tesla P100
4. **Discussion and summary**

Calculate required data size for 1 bucket

- **Data size of neighbour-particle search in MPS**
  1. A bucket access adjacent 27 (=3*3*3) buckets
  2. Maximum number of particles in a bucket is around 33
  3. Particle number and position (x,y,z) must be required
     - ➢ 27 buckets * 33 particles/bucket * (2 B + 4 B * 3) = 12,474 B
  - ✓ Bucket management structure
     - ➢ 72 B * 27 buckets = 1,944 B
  1. Some physical quantities are required in addition

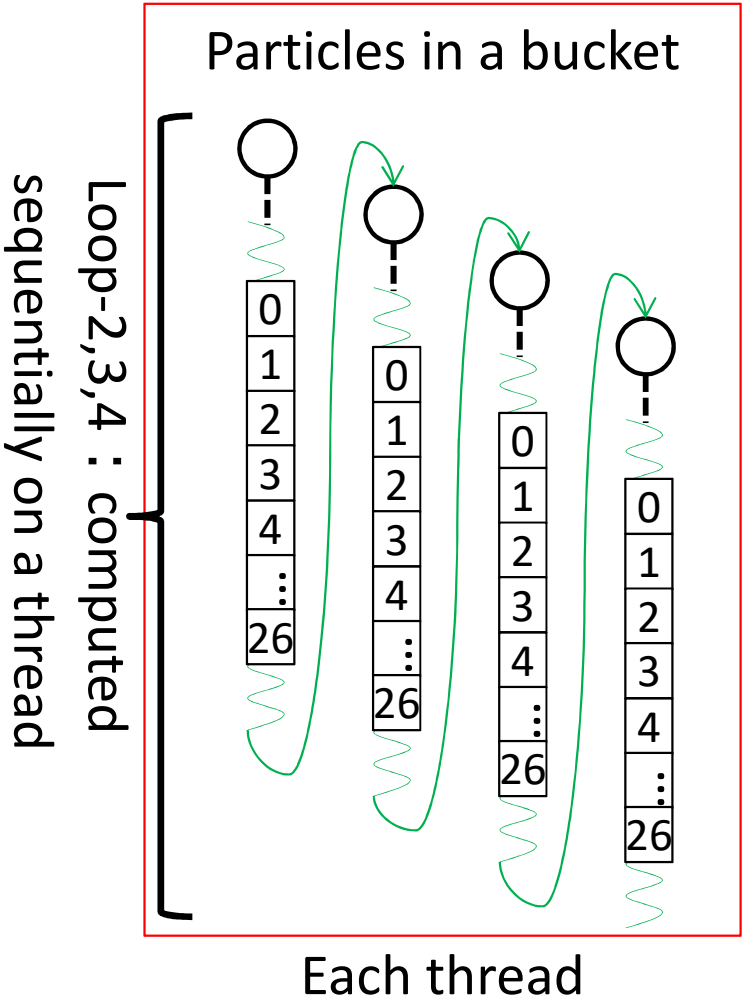


- **For particle density computation**
  - ✓ Density (= one single FP = 4 bytes) is required
     - ➢ 27 buckets * 33 particles/bucket * 4 B = 3,564 B
  - ✓ 17,982 bytes = 17.9 KB required for a bucket computation

If the size of cache for each processing element is larger than 17.9KB, each bucket computation can be done with only cache

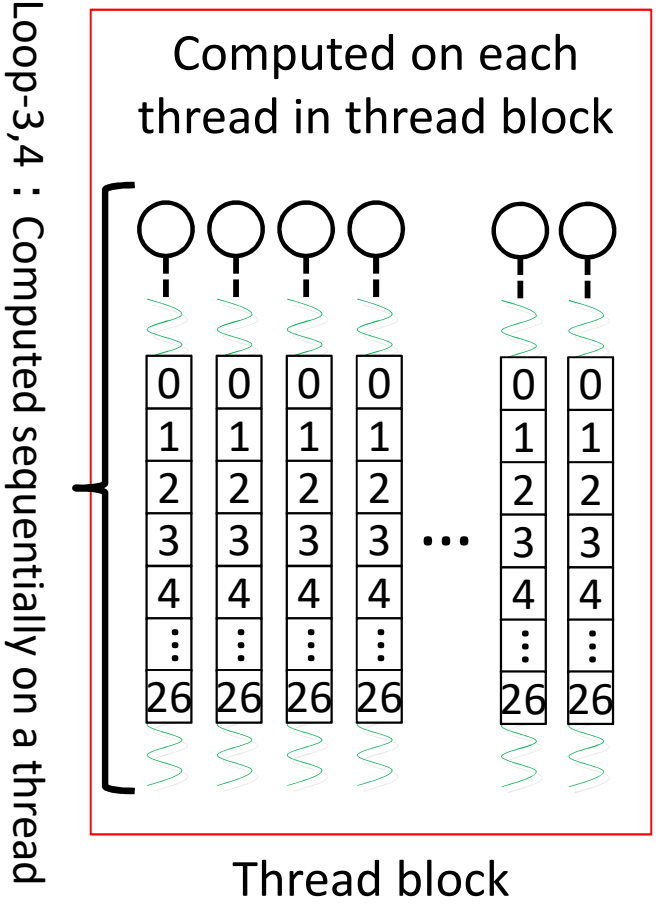
Opt. 1: 1 thread/ 1 bucket

- **One thread computes particles in a bucket**
  - ✓ Loop-1 : Computed by each thread in parallel
  - ✓ Loop-2,3,4 : Computed sequentially by a thread
  - ✓ Each thread requires 17.9KB
  - ✓ Load of each thread will be imbalance
- **For Skylake/KNL**
  - ✓ L1 cache (512KB/32KB) is large enough
  - ✓ Loop-1 : *!$omp parallel do schedule*
- **For P100/V100**
  - ✓ L1 cache (24KB/32t) is not large enough
  - ✓ All the CUDA thread will be activate



Opt. 2: 1 thread block / 1 bucket

- **One CUDA thread block computes a bucket**
  - ✓ Loop-1 : Computed by each thread block in parallel
  - ✓ Loop-2 : Computed by each thread in a thread block
  - ✓ Loop-3,4 : Computed sequentially by a thread
  - ✓ Each thread block requires 17.9KB
- **For Skylake/KNL**
  - ✓ Vectorize inner-loop
  - ✓ Loop-4 : *!$omp simd* (not vectorized)
- **For P100/V100**
  - ✓ L1 cache (24KB) is large enough
  - ✓ All the CUDA thread will not be activate
  - ✓ Loop-1: *!$acc parallel vector length(N), &*
    !$acc loop gang
  - ✓ Loop-2: *!$acc loop vector*



Outline

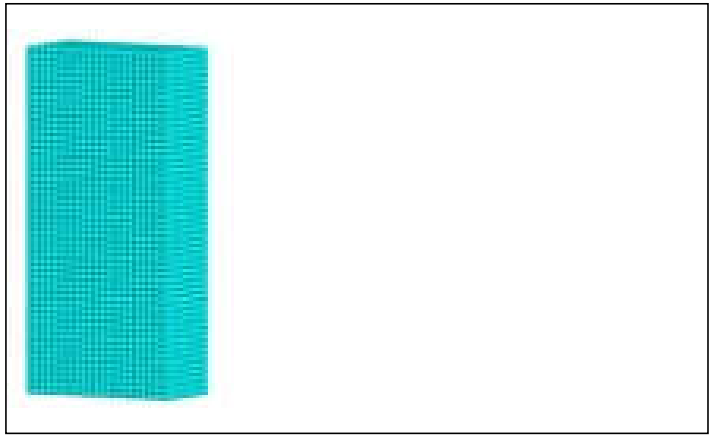
1. ***P-Flow*: JAXA's in-house MPS program**
   i. Moving Particle Simulation (MPS) method
   ii. P-Flow and its internal structure
2. **Proposed optimization techniques**
   i. Calculates required data size for 1 bucket
   ii. Opt. 1: 1 thread/ 1 bucket
   iii. Opt. 2: 1 thread block / 1 bucket
3. **Implementation and evaluation**
   i. Xeon/Xeon Phi
   ii. Tesla P100
   iii. Xeon/Xeon Phi, Tesla P100
4. **Discussion and summary**

Evaluation Environment

- **Evaluation Environment**
  - ✓ 2CPU+2GPU node, except for Xeon Phi KNL 7210

Comp. node	Rpeak single [TFLOPS]	Frequency [GHz]	Num of threads	L1 cache size [KB]	Memory B.W. [GB/s]	Host CPU
Xeon Gold 6510	N/A	2.7	36	512	256	N/A
Xeon Phi KNL 7210	5.3	1.3	256	32	480	N/A
P100 (PCIe)	9.3	1.1	3,584	24	732	Xeon Gold 6510
P100 (NVlink)	10.6	1.3	3,584	24	732	Xeon E5-2695v4

- **Dataset**
  - ✓ Dam break problem: 40cm×40cm×8cm
  - ✓ 70x70x14 buckets, 2,247,750 particles
  - ✓ Average first 200 timesteps
  - ✓ Time measurement: MPI_Wtime()



Dam break problem at 1st timestep

Compilers

- **Xeon / Xeon Phi: Intel Fortran compiler ver. 17.0.4**
  - ✓ Compiler option:
    "-qopenmp -O3 -xCOMMON-AVX512 -fp-model fast=2"
  - ✓ Affinity of thread: "granularity=fine,compact"
  - ✓ Memory mode of Xeon Phi:
    Flat-Quadrant ($ numactl -membind=1)
  - ✓ MPI Library: IntelMPI 5.0.2
  - ✓ Enable Hyper Threading: Xeon=2th/Core, Xeon Phi=4th/Core
- **Tesla P100, V100: PGI Fortran compiler ver. 17.7**
  - ✓ Compiler option: "-acc -ta=nvidia:cc60, fastmath"
  - ✓ MPI Library : OpenMPI 1.10.5

Xeon Gold 6150 / Xeon Phi KNL 7210

- **Eval 1: Scheduling policy of OpenMP**
  - ✓ "dynamic" is faster than "static": 14.8%@Xeon, 36.1%@KNL
  - ✓ Xeon is more sensitive in terms of load balancing
- **Eval 2: Flat or Hybrid parallelization ?**
  - ✓ Xeon : "Hybrid+dynamic" and "Hybrid+static" are 36% and 3.9% faster than "Flat", respectively.
  - ✓ KNL : Almost the same in all cases.
    - ➢ Flat (only MPI parallelization) works almost the same performance as Hybrid 3 (dynamic). Both decompose the domain in similar manner?

Parallelization	Hybrid 1	Hybrid 2	Hybrid 3	Flat
Policy	dynamic	static	guided	N/A
Num of processes	2	2	2	72 or 256
Num of threads	36	36	36	N/A
KNL 7210 (1 node)	52.1 (104.1)	58.6 (117.2)	60.6(121.1)	52.7(105.4)
Gold 6150	35.1	46	47.8	47.8

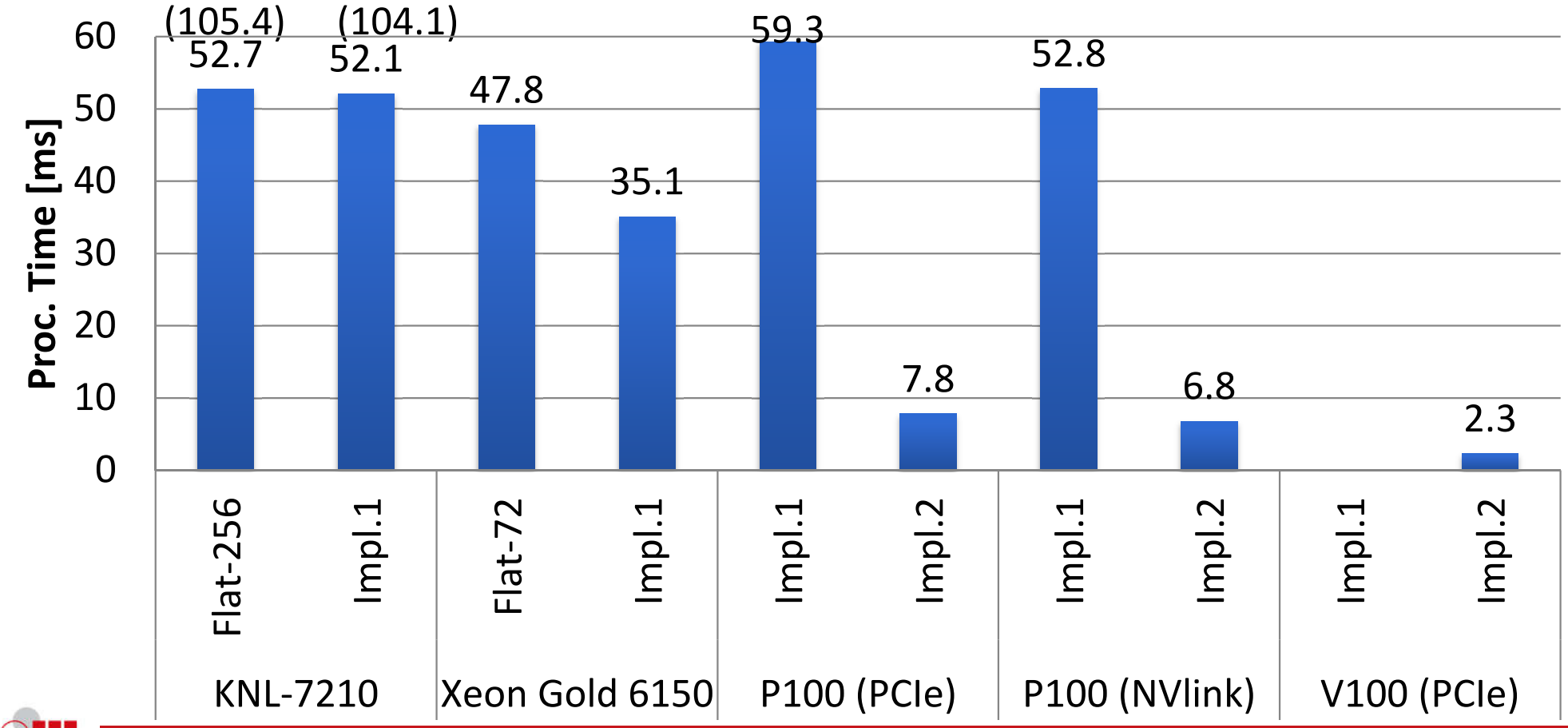
Tesla P100 (PCIe, NVlink), V100

- **Eval 1: Comparison between Opt.1 and Opt.2**
  - ✓ Opt 1 : Each CUDA thread works differently
  - ✓ Opt 2 : Works well. Seven times faster than Opt 1
- **Eval 2: Block size**
  - ✓ Block size 64 is 0.2% faster than 32. (Unexpected result.)
    - ➢ The number of average bucket is around 10. We expected that block size 32 is the fastest.
  - ✓ Block size 128 performs not bad.
    - ➢ The number of used register is 72. Is this the reason?

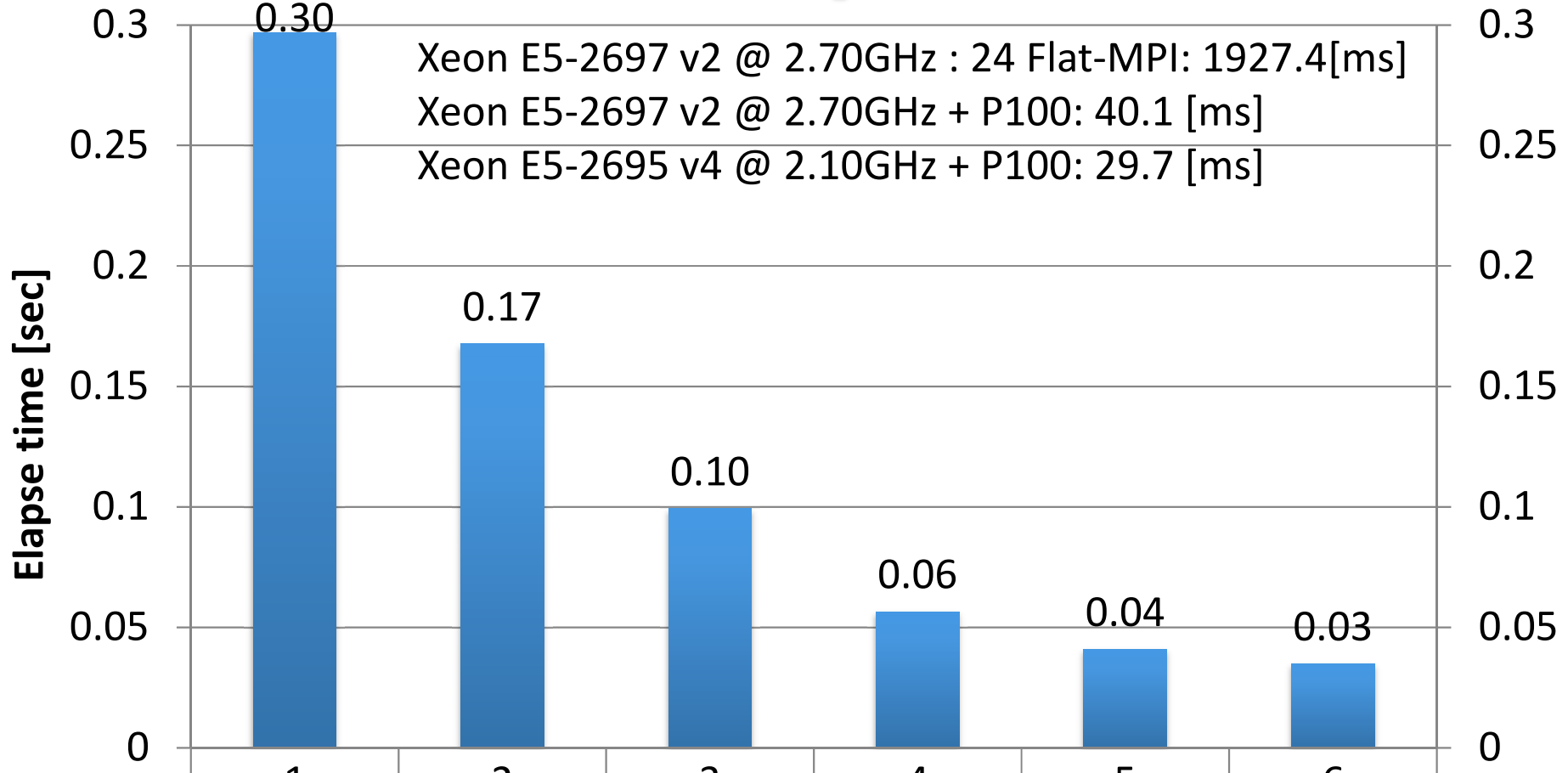
Optimization	Opt. 1	Opt. 2			
Block size	64	16	32	64	128
P100 (PCIe)	59.3	13.4	7.82	7.80	8.3
P100 (NVlink)	52.8	11.6	6.79	6.77	7.2
V100 (PCIe)	?	?	2.3	?	?

Xeon/KNL, Tesla P100/V100

- **V100 (NVLink) is the fastest.**
  - ✓ x15.2 and x22.7 times faster then Xeon and KNL, respectively.
- **Vectorization is not done and not fit for Xeon and KNL.**
  - ✓ Xeon and KNL are used just as a multi-thread machine



Strong scaling on 2~64 GPUs on Reedbush-H @Univ. Tokyo

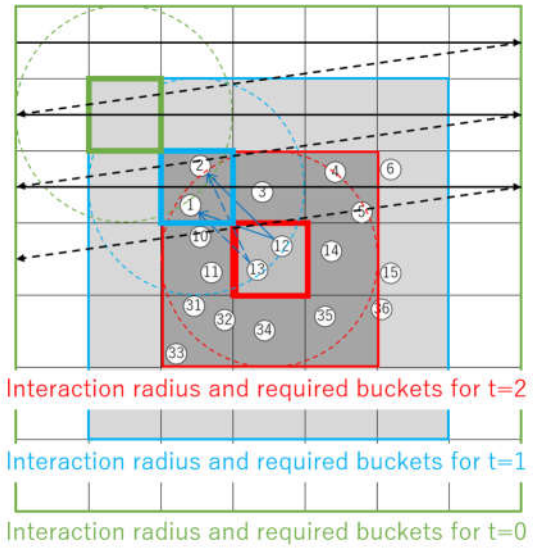


	1	2	3	4	5	6
Node	1	2	4	8	16	32
GPU	2	4	8	16	32	64
Particles / GPU	1123875	524475	261738	149850	65934	3996
Laplacian_u	0.297025	0.167592	0.0992125	0.0565129	0.0408499	0.0348665

What's next and requests

【Next optimization】

- **Apply temporal blocking**
  - ✓ Temporal Blocking is gathering attention in stencil computation
  - ✓ Reduce the number of access to external memory
  - ✓ Pre-load additional adjacent bucket



【Requests】

- **Improvement of functionality of *!$acc cache* directive**
  - ✓ Hard to understand its behavior
- **Improvement of usability of CUDA-aware MPI with OpenACC**
  - ✓ How do we check if the CUDA -aware MPI w/ OpenACC works correctly?

  → T.Daniel@PGI answered me
- **Commit to GCC**

Short Summary

- ***P-Flow* : JAXA's in-house MPS program**
  - ✓ Apply MPS method to aerospace field
  - ✓ Neighbour-particle search is a bottle neck
- **Propose two optimizations while considering data size for 1 bucket**
  - ✓ 1 bucket requires 17.9 KB at maximum
  - ✓ Opt. 1: 1 thread/ 1 bucket
  - ✓ Opt. 2: 1 thread block / 1 bucket
- **Evaluate on Xeon / Xeon Phi / P100**
  - ✓ Intel Xeon Skylake-SP : 104.1[ms]
  - ✓ Intel Xeon Phi KNL : 35.1[ms]
  - ✓ NVIDIA Tesla P100(NVlink) : 6.8[ms]
  - ✓ NVIDIA Tesla P100(PCIe) : 7.8[ms]
  - ✓ NVIDIA Tesla V100(PCIe) : 2.3[ms]