

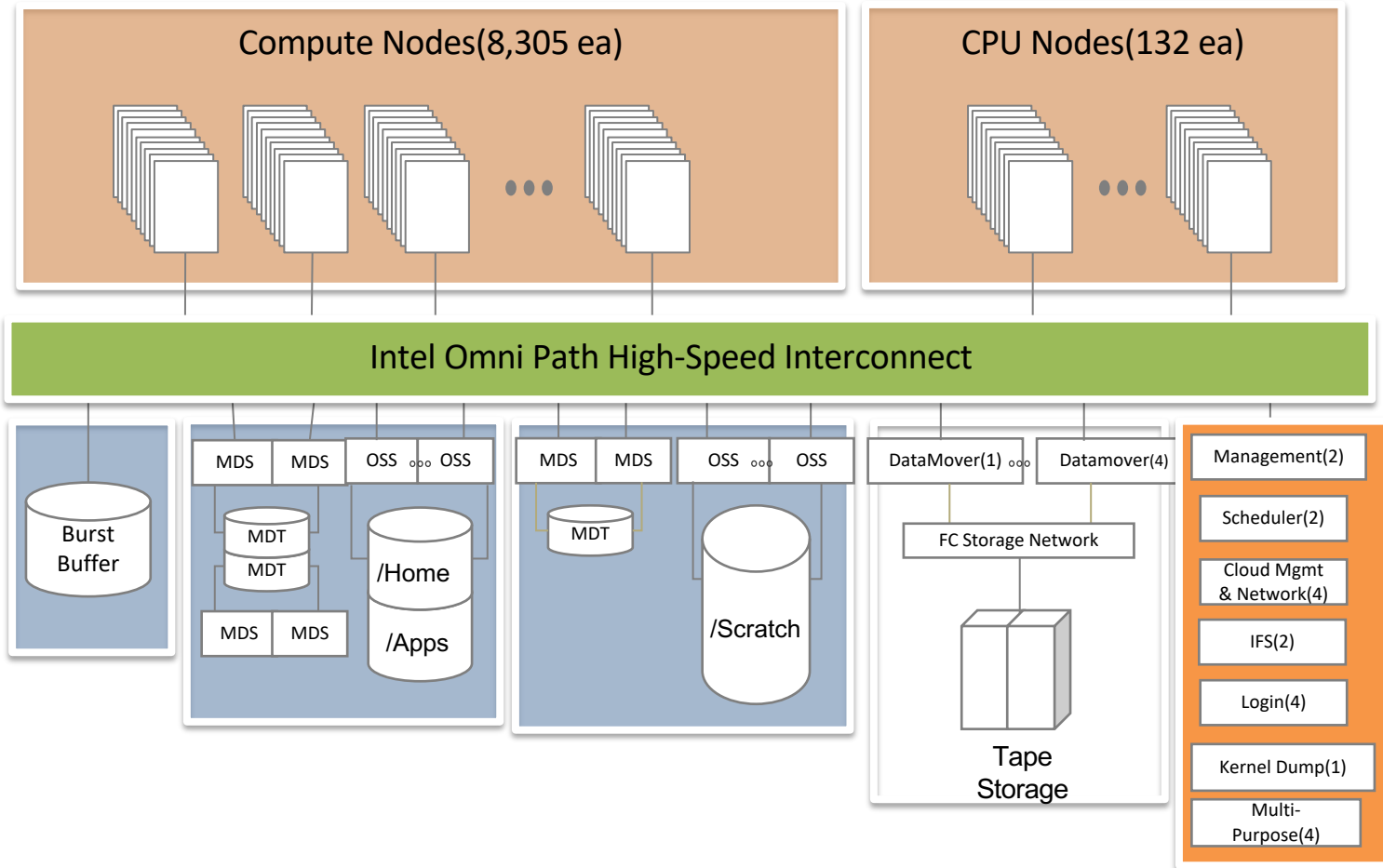


# 슈퍼컴퓨터 5호기 누리온 소개

2021년 6월 17일

권오경  
슈퍼컴퓨팅응용센터  
국가슈퍼컴퓨팅본부

## Main System





Theoretical performance(Rpeak) 25.7PF = 25.3PF CS400 w/KNL+0.4PF CS500 w/SKL  
Measured performance(Rmax) 13.9PF (54.2% of RPeak)

## Storage

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



## Computing nodes

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



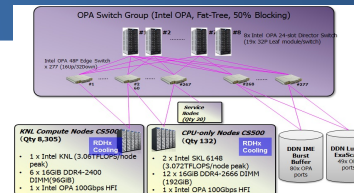
## CPU-only nodes

Cray 3111-BA000T(2U enclosure), 132 SKL Compute modules




## Interconnect

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



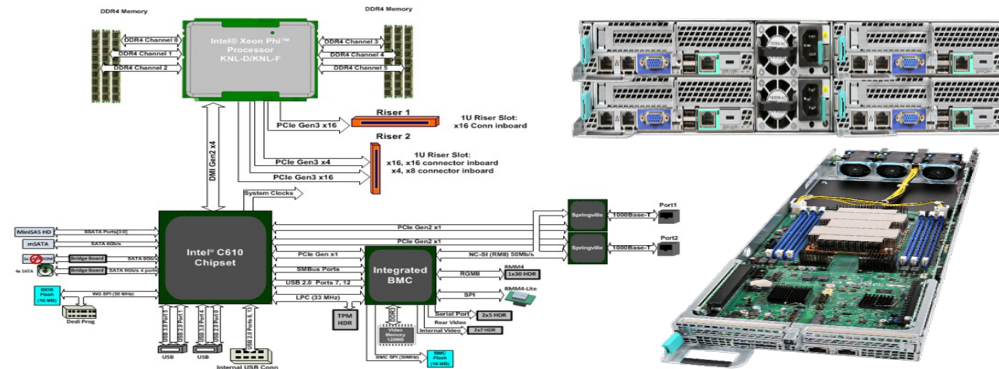
KISTI-5 project

 Theoretical performance(Rpeak) 25.7PF = 25.3PF CS400 w/KNL+0.4PF CS500 w/SKL  
Measured performance(Rmax) 13.9PF (54.2% of RPeak)

## Computing nodes

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

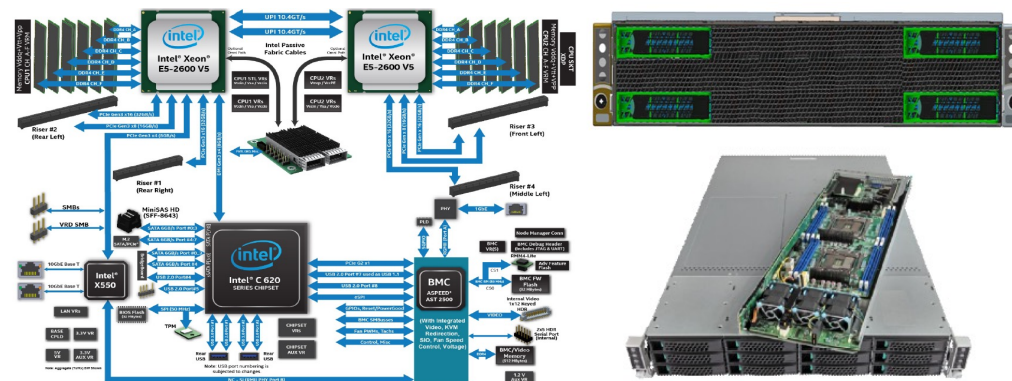
- 1x Intel Xeon Phi KNL 7250 processor
- 96GB (6x 16GB) DDR4-2400 RAM
- 1x Single-port 100Gbps OPA HFI card
- 1x On-board GigE (RJ45) port



## CPU-only nodes

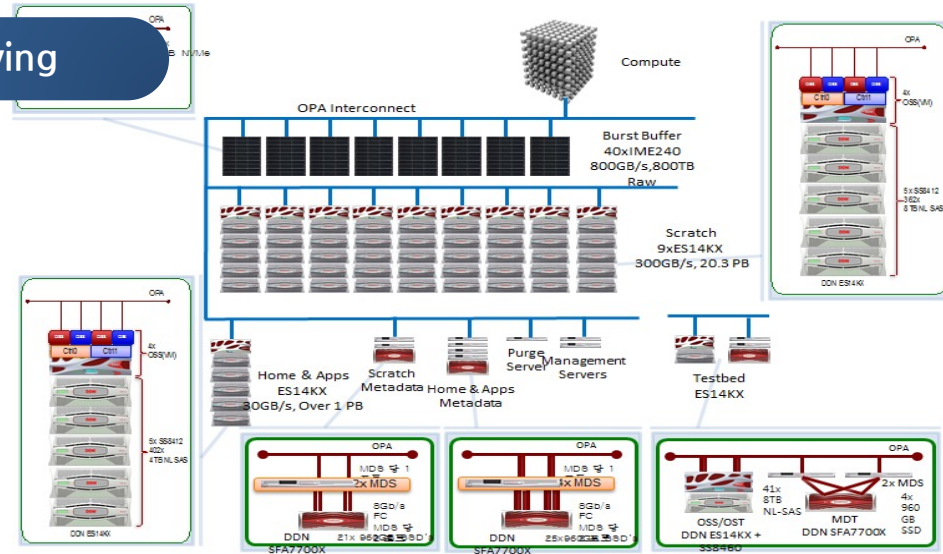
Cray 3111-BA000T(2U enclosure), 132 Skylake Computing modules

- 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



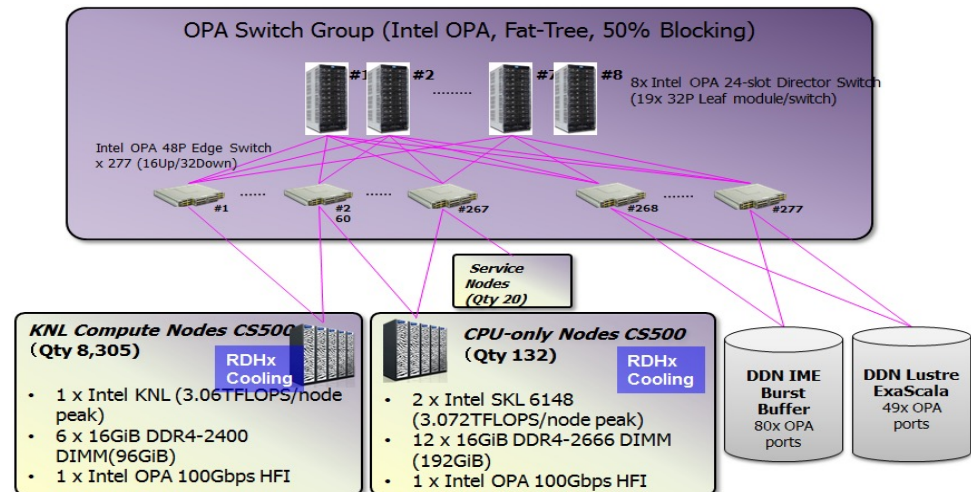
## Storage 20PB SFS@300GB/s, 10PB Archiving

- Global scratch: 20PB, 0.3TB/s  
(DDN ES14KX 9ea, 360 x 8TB disk each)
- Home and application directory 1PB  
(IME240 48ea 19 NVMe SSD each)
- NVMe Burst Buffer: 0.8PB, 0.8TB/s  
(IME240 48ea 19 NVMe SSD each)
- Cray TSMFS and IBM TS4500



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

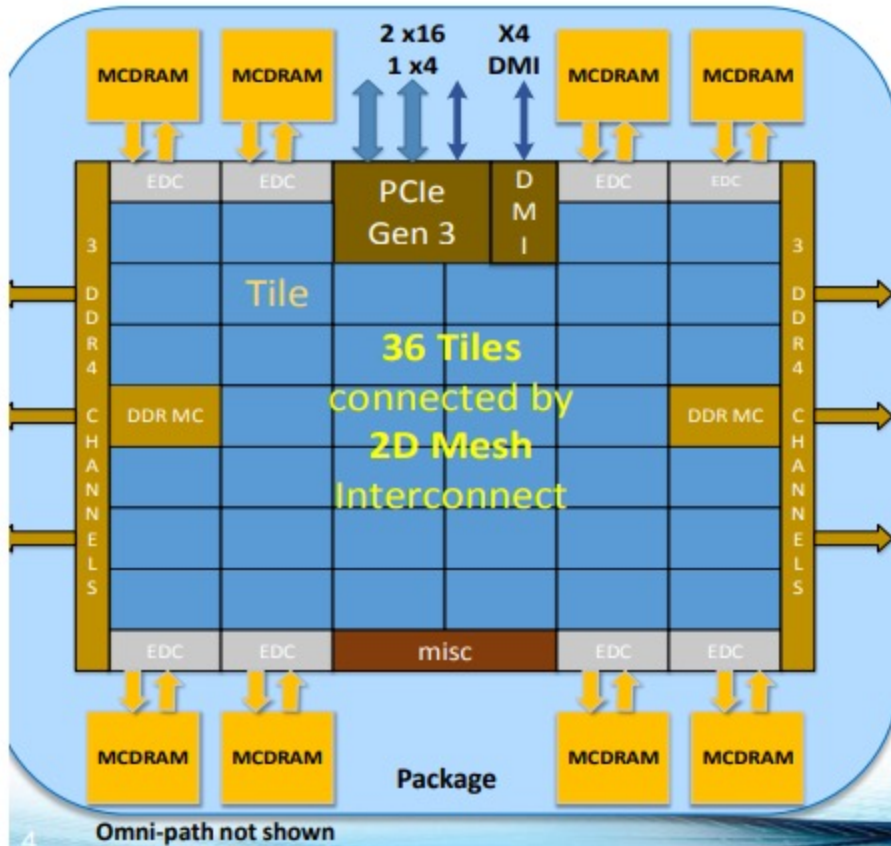
- Intel OPA High-speed interconnect switch  
274x 48-port OPA edge switches  
8x 768-port OPA core switches
- Bandwidth: 12.3 GB/sec
- Bisectonal Bandwidth : 27 TB/sec
- $10^{-16}$  BER(Bit Error Rate), Adaptive routing





- **KNL (Xeon Phi 7250)**
  - 68 physical cores/socket
  - 272 logical cores/socket
  - 1.4 GHz (up to 1.6Ghz in Turbo mode)
  - 32 double precision operations per cycle (16 DP FLOPS per cycle per AVX-512 FMA unit)
  - 16 GB of fast memory (MCDRAM) 96GB of DDR4 DRAM
  - ~115 GB/s DDR4 bandwidth
  - MCDRAM has ~ 5x DDR4 bandwidth
  - Cache: L1 2.3MB L2 34MB
- **SKL (Xeon SKL 6148)**
  - 20 physical cores/socket
  - 40 logical cores/socket
  - 2.4 GHz (up to 3.7Ghz in Turbo mode)
  - 32 double precision operations per cycle
  - 4.9 GB DRAM/core (196 GB/node)
  - ~119 GB/s Memory Bandwidth
  - Cache: L1 1.25 MB, L2 20MB, L3 27.5MB

## Knights Landing Overview



### TILE

2 VPU	CHA	2 VPU
Core	1MB L2	Core

**Chip: 36 Tiles** interconnected by 2D Mesh

**Tile: 2 Cores + 2 VPU/core + 1 MB L2**

**Memory: MCDRAM: 16 GB** on-package; High BW

**DDR4: 6 channels @ 2400** up to 384GB

**IO: 36 lanes** PCIe Gen3. 4 lanes of DMI for chipset

**Node: 1-Socket** only

**Fabric: Omni-Path** on-package (not shown)

**Vector Peak Perf: 3+TF DP** and 6+TF SP Flops

**Scalar Perf: ~3x** over Knights Corner

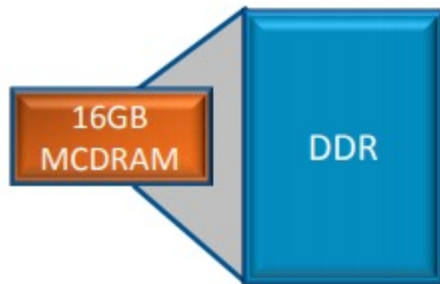
**Streams Triad (GB/s): MCDRAM : 400+; DDR: 90+**

Source Intel: All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. KNL data are preliminary based on current expectations and are subject to change without notice. 1.Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX). 2.Bandwidth numbers are based on STREAM-like memory access pattern when MCDRAM used as fast memory. Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software configuration may affect actual performance.

## Memory Modes

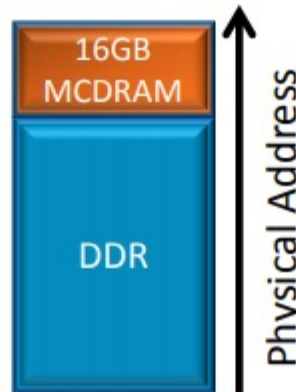
Three Modes. Selected at boot

### Cache Mode



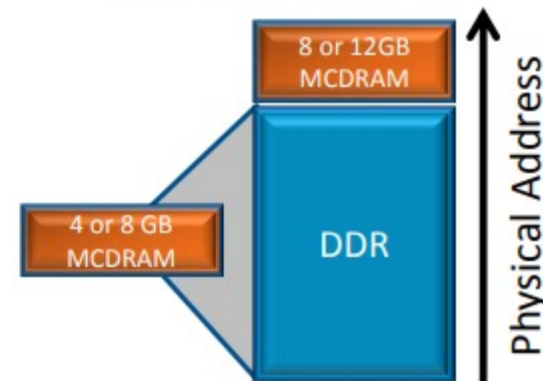
- SW-Transparent, Mem-side cache
- Direct mapped. 64B lines.
- Tags part of line
- Covers whole DDR range

### Flat Mode



- MCDRAM as regular memory
- SW-Managed
- Same address space

### Hybrid Mode



- Part cache, Part memory
- 25% or 50% cache
- Benefits of both



## Flat MCDRAM: SW Architecture

**MCDRAM exposed as a separate NUMA node**



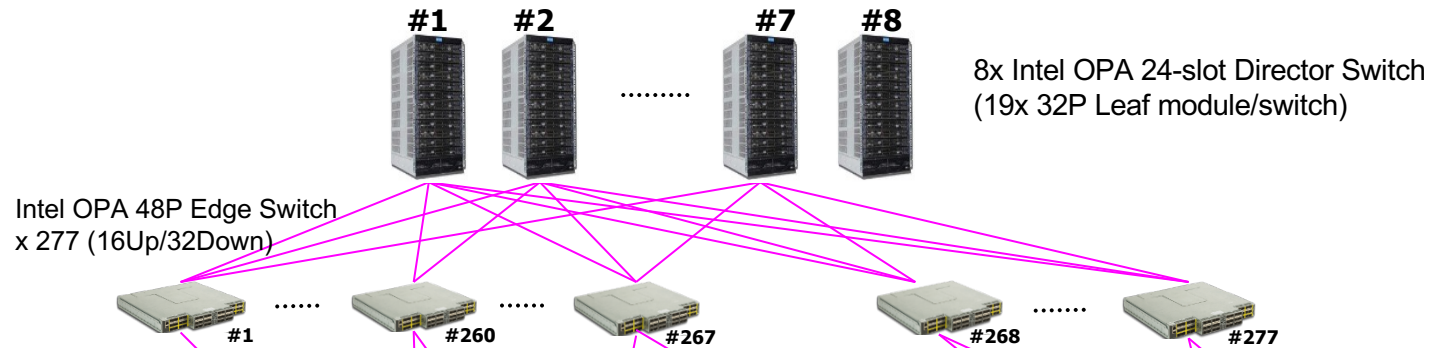
Memory allocated in DDR by default → Keeps non-critical data out of MCDRAM.

Apps explicitly allocate critical data in MCDRAM. Using two methods:

- **“Fast Malloc”** functions in High BW library (<https://github.com/memkind>)
  - Built on top to existing *libnuma* API
- **“FASTMEM”** Compiler Annotation for Intel Fortran

**Flat MCDRAM with existing NUMA support in Legacy OS**

## OPA Switch Group (Intel OPA, Fat-Tree, 50% Blocking)



Service Nodes  
(Qty 23)

### KNL Compute Nodes CS500 (Qty 8,305=25.3PF)



- 1x Intel KNL 7250 (3.06TFLOPS/node peak)
- 6x 16GiB DDR4-2400 DIMM(96GiB)
- 1x Intel OPA 100Gbps HFI

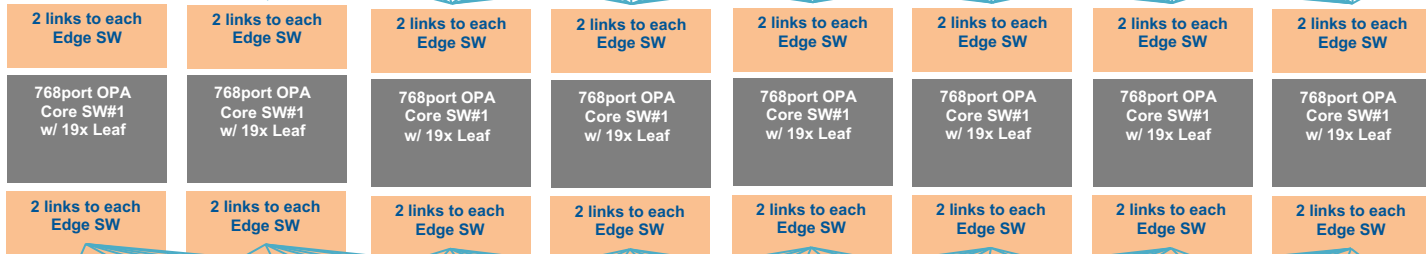
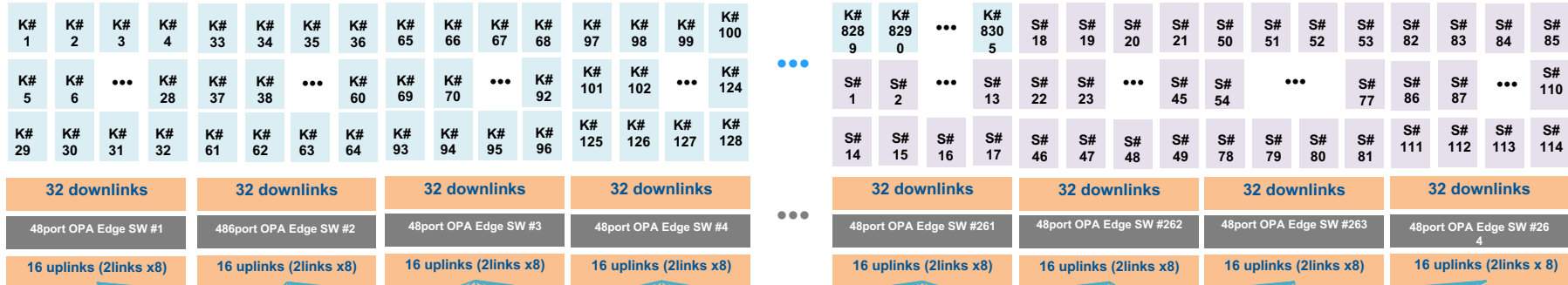
### CPU-only Nodes CS500 (Qty 132=0.4PF)



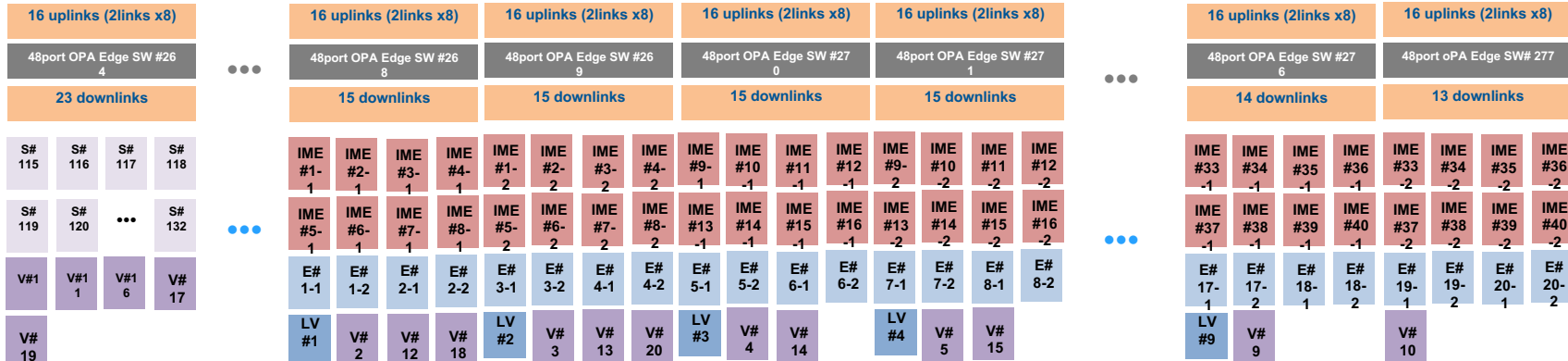
- 2x Intel SKL 6148 (3.072TFLOPS/node peak)
- 12x 16GiB DDR4-2666 DIMM(192GiB)
- 1x Intel OPA 100Gbps HFI

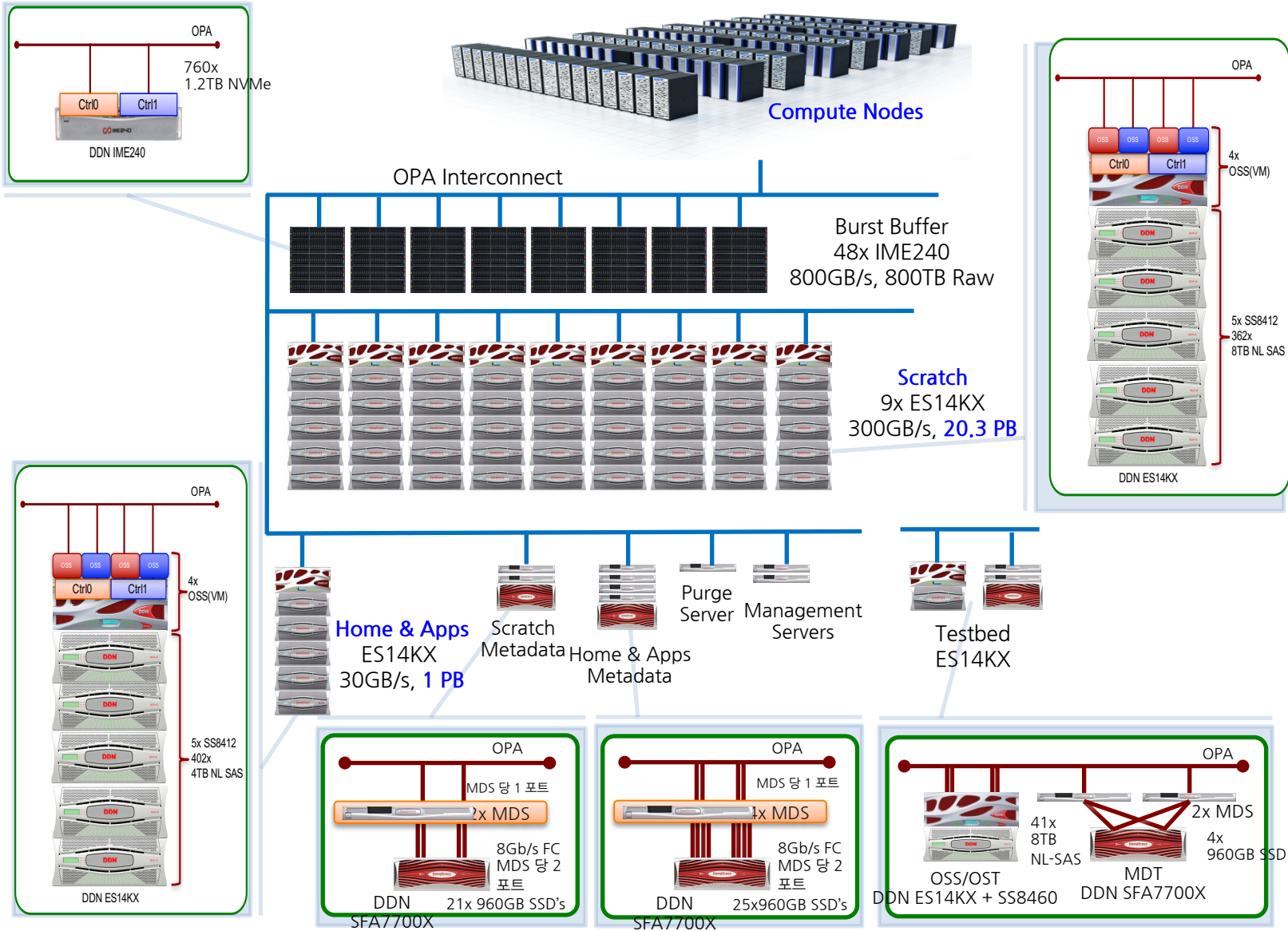
DDN IME  
Burst Buffer  
80x OPA  
ports

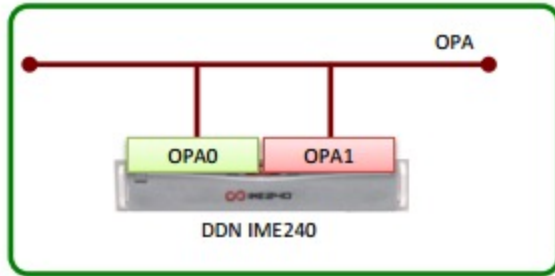
DDN Lustre  
ExaScaler  
49x OPA  
ports



K: KNL Compute Node (x 8305)  
S: SKL CPU-Only Node (x 132)  
V: Service Node (x23)  
IME: IME Node (x40, 2x HCA/node)  
LV: Storage Service Node (x 9)  
E: ExaScaler Storage (x 40)



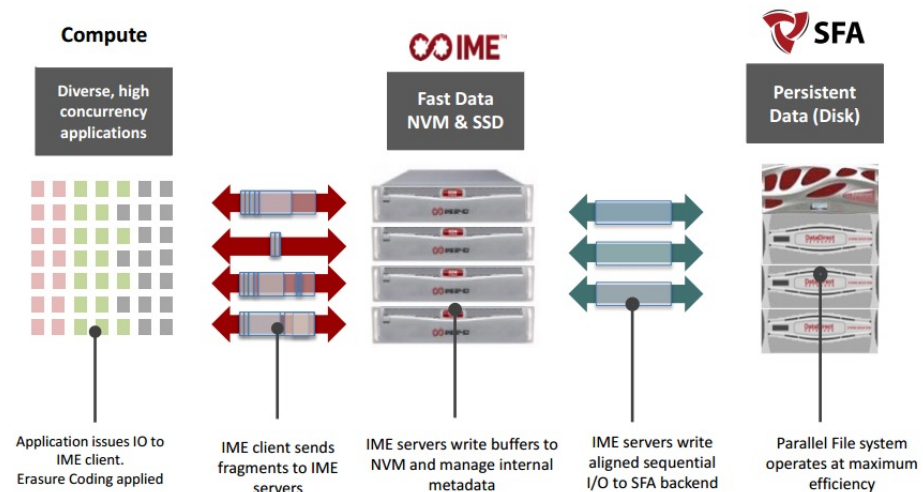


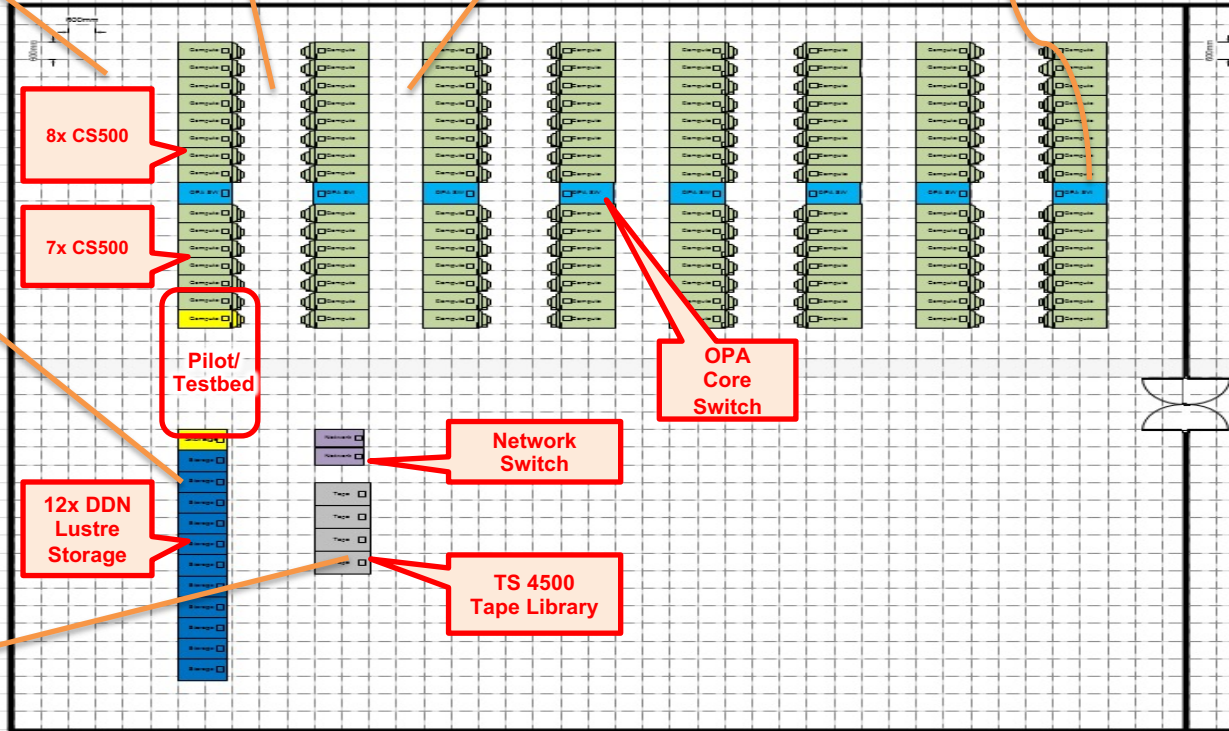
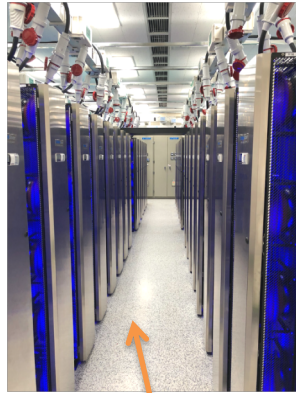


주요 구성 요소	내역
시스템	DDN IME240, Infinite Memory Engine Appliance
소프트웨어 버전	CentOS 7.3, IME 1.2
최대 IO 성능	20 GB/s
네트워크 인터페이스	2x OPA
SSD 타입	1.2TB, NVMe 드라이브
SSD 수량	16ea(1.2TB NVMe)+1ea(450GB SSD)
용량(RAW)	19.2TB
전력량(W,Nominal)	1,028
발열량(BTU/hr,Nominal)	3,506

주요 구성 요소	내역
총 시스템 수	48 x IME240
총 용량(Raw)	921.6TB
총 용량(EC, 15+2적용 시)	813TB
총 최대 IO 성능	0.8TB/s
총 전력량(W,Nominal)	41,120
총 발열량(BTU/hr,Nominal)	140,307

- New Cache Layer using NVMe SSDs inserted between compute cluster and PFS
  - IME is configured as CLUSTER with multiple SSD servers
  - All compute nodes can access cache data on IME
- Accelerates "bad" IO on PFS
- Accelerates small IO or random IO by high IOPS due to SSD and IO management
  - PFS is pretty good at large sequential IO
- Can be configured as cache layer having huge IO bandwidth







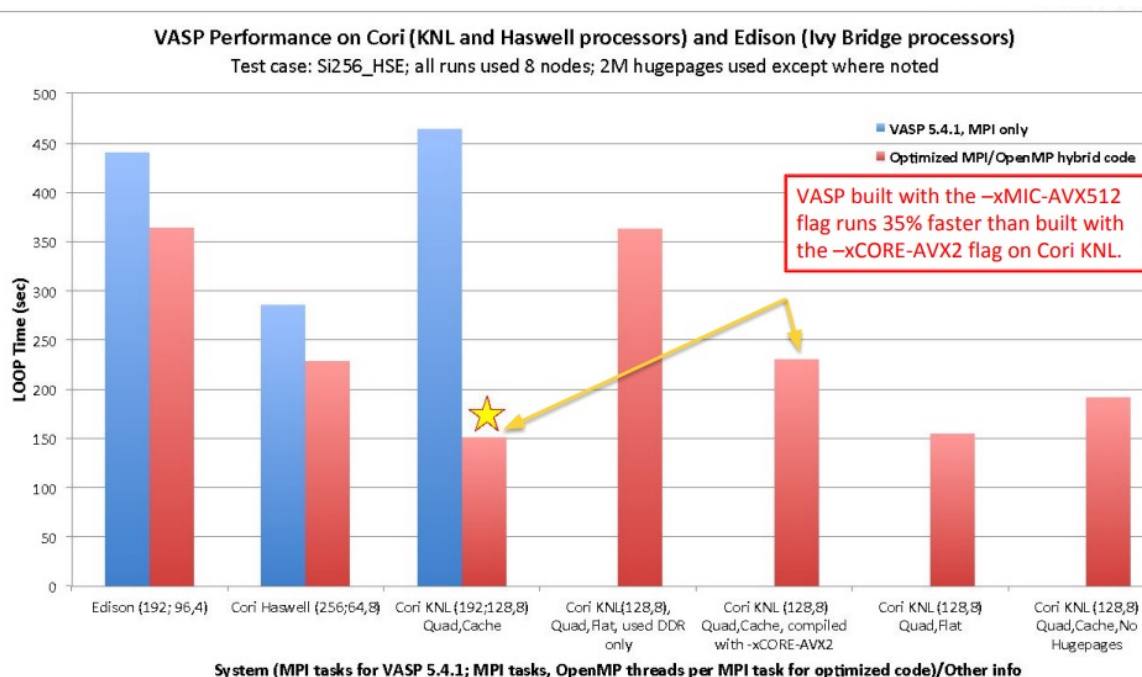
Category	Sub-category	Compute Nodes	CPU-only Nodes
Model	System Model	Cray CS500	Cray CS500
	Number of Nodes	8,305	132
	Number of Racks	116	2
Performance	Peak Performance	25.3PFLOPS	0.4PFLOPS
CPU Type	Model	Intel Xeon Phi 7250 (KNL)	Intel Xeon 6148 (Skylake)
	Peak Performance	3.0464TFLOPS	1.536TFLOPS
	# of Cores per CPU	68	20
	# of CPUs per Node	1	2
	On-package Memory	16GB, 490GB/s	-
HCA	Model	Intel 100HFA016LS	Intel 100HFA016LS
Main Memory (Off-package)	Model	16GB DDR4-2400	8GB DDR4-2666
	Configuration	16GB x6, 6Ch per CPU	8GB x12, 6Ch per CPU
	Size per Node(GB)	96GB	192GB
	Bandwidth	115.2GB/s	128GB/s
	Total Size	778.5TB	24.76TB
Local Disk	Capacity	N/A	1TB
High Performance Interconnect	Topology	FatTree	
	Blocking Ratio	50% Blocking	
	# of Switches	274x 48-port OPA edge switches 8x 768-port OPA core switches	
	Bandwidth per Port	12.3GB/sec	
	Bisectional Bandwidth	27,060GB/sec	
Burst Buffer	# of Servers	DDN IME240 Servers 40ea	
	Configurations of Disks	19x 1.2TB NVMe SSD, 2x 0.45TB NVMe SSD	
	Total Capacity	0.8PB usable	
	Bandwidth	20GB/sec per server, 0.8TB/s total	
Parallel File System	Scratch MDS	DDN SFA7700X 1ea, MDS 2ea	
	Home/Apps MDS	DDN SFA7700X 1ea, MDS 4ea	
	Scratch OSS	4x OSS(embedded) per ES14KX DDN ES14KX 9ea w/ 360x 8TB each	
	Home/Apps OSS	4x OSS(embedded), DDN ES14KX 1ea w/ 400x 4TB	
	Total Capacity	20PB usable	
	Bandwidth	0.3TB/s	



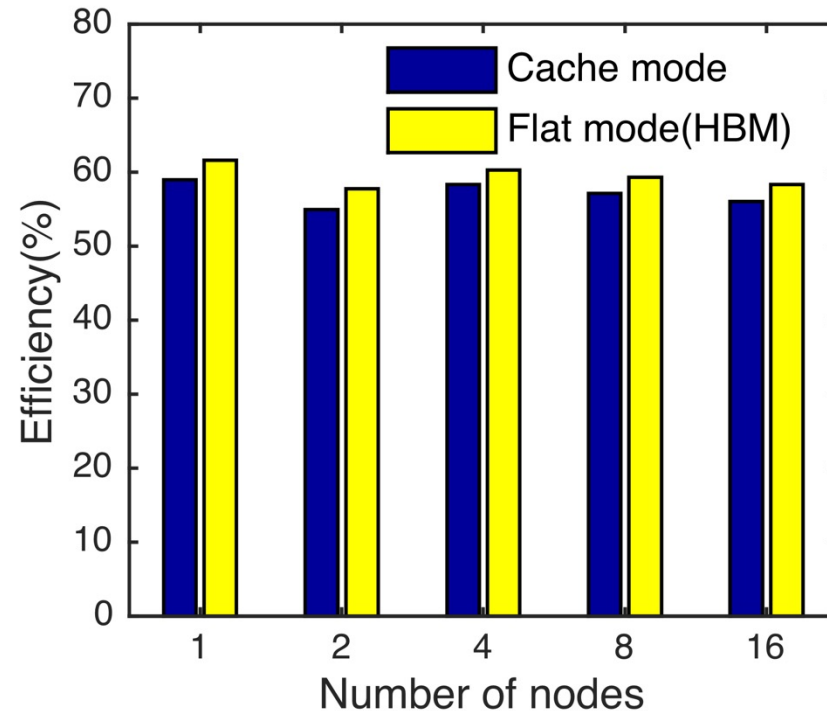
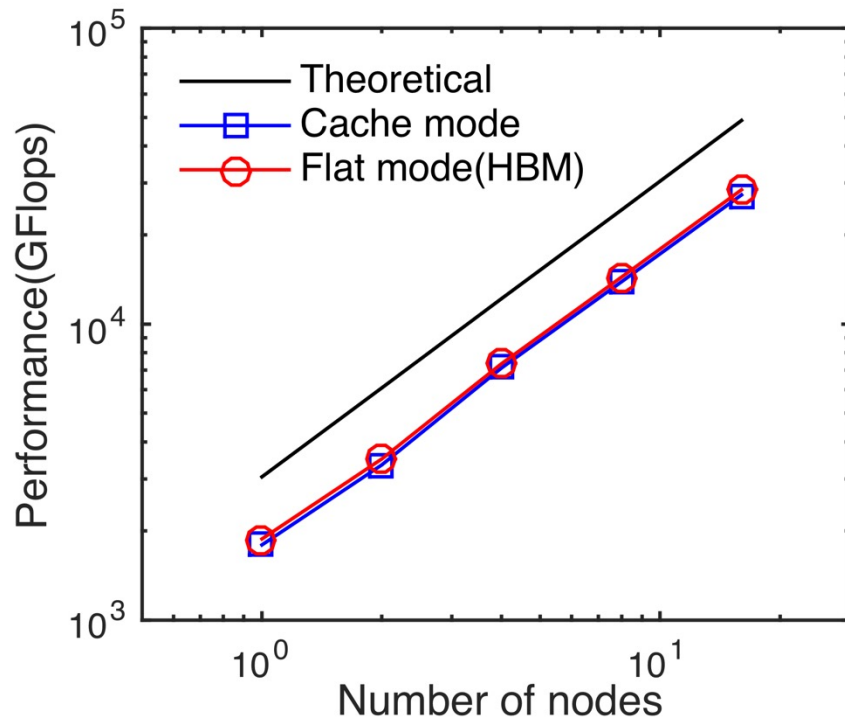
## • Compiler wrappers

	Intel	GNU	Cray	Module
KNL	-xMIC-AVX512	-march=knl	-h cpu=mic-knl	<b>craype-mic-knl</b>
Skylake	-xCORE-AVX512	-march=skylake	-h cpu=skylake	<b>craype-x86-skylake</b>

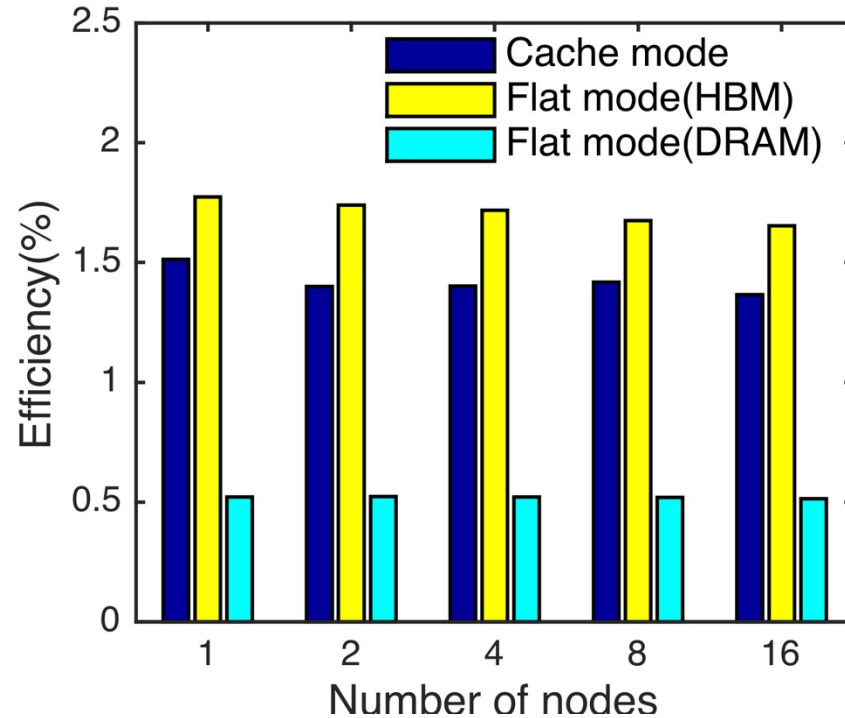
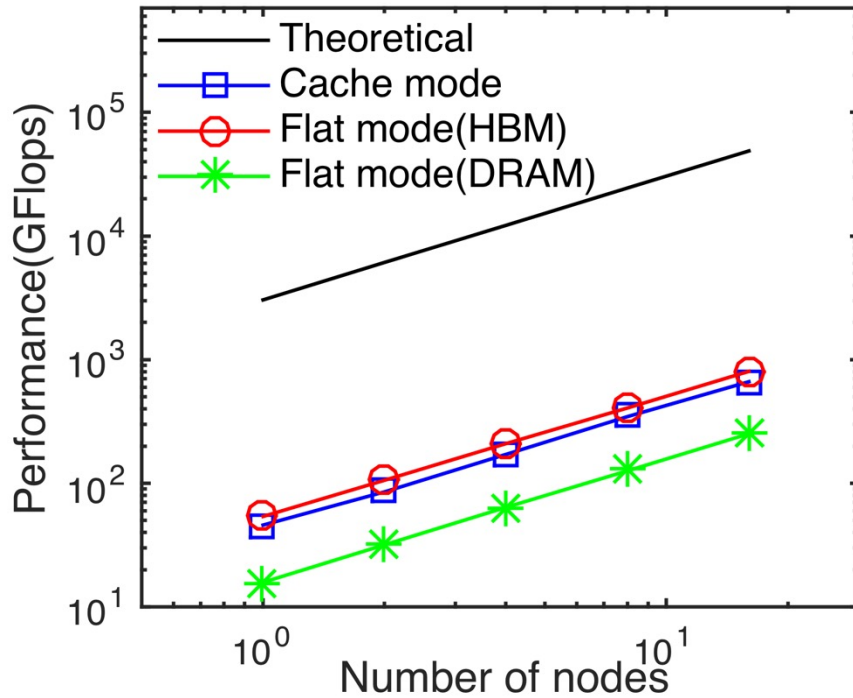
- Intel - better chance of getting processor specific optimizations, especially for KNL
- Cray compiler – many new features and optimizations, especially with Fortran
- GNU - widely used by open software



**Building your application separately for each platform could be important to get optimal performance.**



- Cache mode and flat mode with HBM show almost same performance
- 1.8TFlops / node is obtained for both memory modes
- Efficiency, the ratio of measured performance to theoretical performance is ~60% up to 16 nodes, but is ~53% to total number of nodes of Nurion.



- HPCG performance is much lower than theoretical performance: ~51GFlops / node
- Efficiency, the ratio of measured performance to theoretical performance is ~1.7% up to 16 nodes, but 1.5% up to total number of nodes of Nurion.
- Cache mode and flat mode show quite large performance difference since HPCG highly depends memory bandwidth.

# 슈퍼컴에서 최적병렬화 사례 소개



권오경  
슈퍼컴퓨팅응용센터  
국가슈퍼컴퓨팅본부



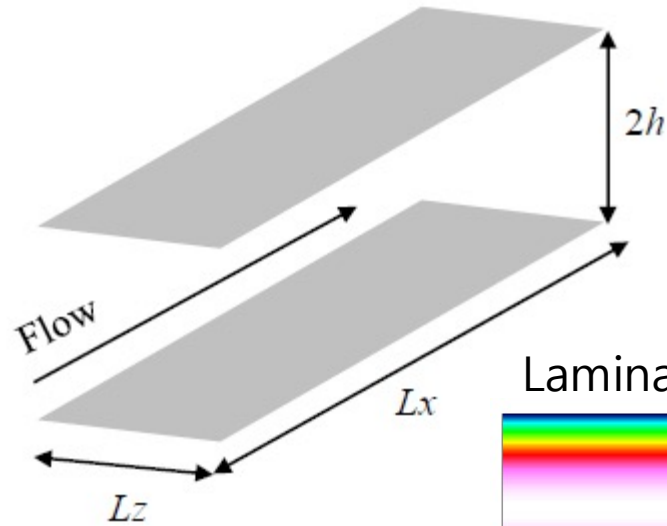
1. MPI Parallel Implementation for Pseudo-Spectral Simulations for Turbulent Channel Flow @ 5호기 Nurion
2. LES를 이용한 축류팬 주위 유동 해석 코드 개발 @ 4호기 Tachyon
3. LES를 이용한 유체-고체 연성 해석을 통한  $Re \geq 10,000$  난류 영역에서의 타코마 다리 붕괴 시뮬레이션 @ 5호기 Nurion



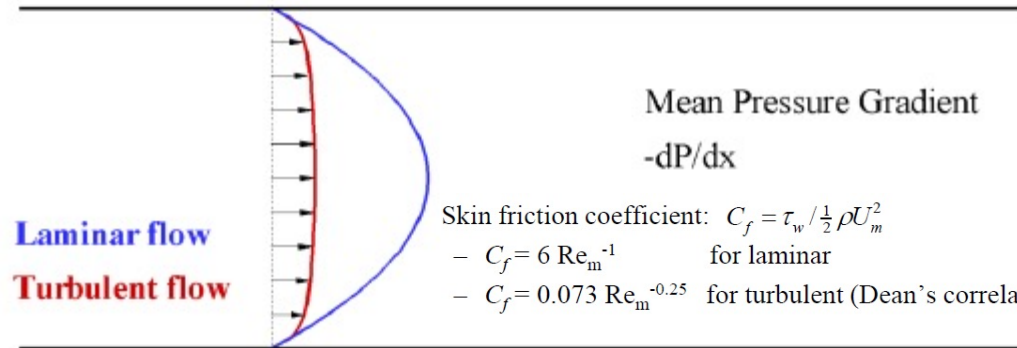
# MPI Parallel Implementation for Pseudo-Spectral Simulations for Turbulent Channel Flow @ 5호기 Nurion



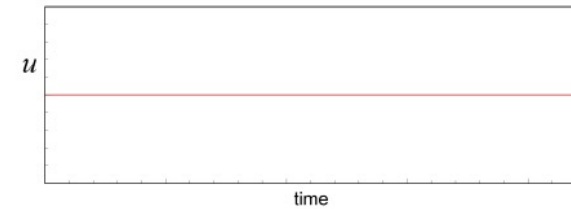
권오경  
슈퍼컴퓨팅응용센터  
국가슈퍼컴퓨팅본부



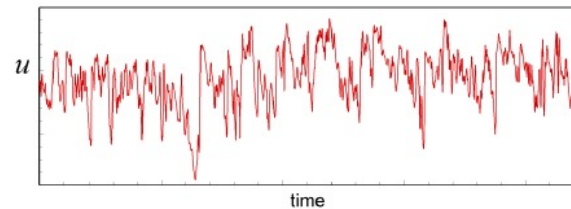
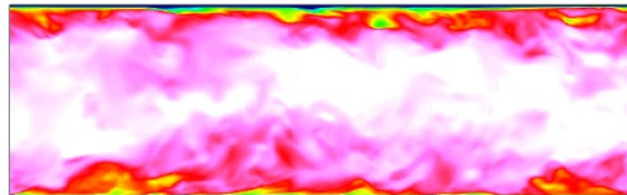
## Velocity profile



Laminar flow → Flow direction



Turbulent flow



$$0 = -\frac{1}{\rho} \frac{dP}{dx} + \frac{d}{dy} \left( \underbrace{-\overline{u'v'}}_{\text{Reynolds shear stress}} + \underbrace{\nu \frac{dU}{dy}}_{\text{Viscous shear stress}} \right)$$



## Governing Equation

The non-dimensionalized incompressible Navier-Stokes equations :

- $$\frac{\partial u_i}{\partial t} = -\frac{\partial p}{\partial x_i} + \epsilon_{ijk} u_j (\omega_k + 2\Omega_k) - \frac{\partial}{\partial x_i} \left( \frac{1}{2} u_j u_j \right) + \frac{1}{R} \nabla^2 u_i, \quad (1)$$

- $$\frac{\partial u_i}{\partial x_i} = 0. \quad (2)$$

Taking the divergence of equation (1) yields a Poisson equation for  $p$  :

- $$\nabla^2 p = \frac{\partial H_i}{\partial x_i} - \nabla^2 \left( \frac{1}{2} u_j u_j \right), \quad (3)$$

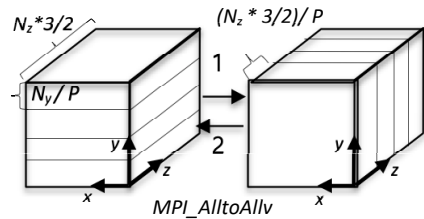
- $$H_i = \epsilon_{ijk} u_j (\omega_k + 2\Omega_k). \quad (4)$$

The Laplacian of equation (1) and using equations (2) and (3) lead to the following equation :

- $$\frac{\partial \nabla^2 u_2}{\partial t} = \left( \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_3^2} \right) H_2 - \frac{\partial}{\partial x_2} \left( \frac{\partial H_1}{\partial x_1} + \frac{\partial H_3}{\partial x_3} \right) + \frac{1}{R} \nabla^4 u_2. \quad (5)$$



## Schematics of domain decomposition methods.



$N_x$ ,  $N_y$  and  $N_z$  : the number of grid points along the X-axis, Y-axis and Z-axis, respectively  
 $\rightarrow$  : the sequence of communication per an iteration

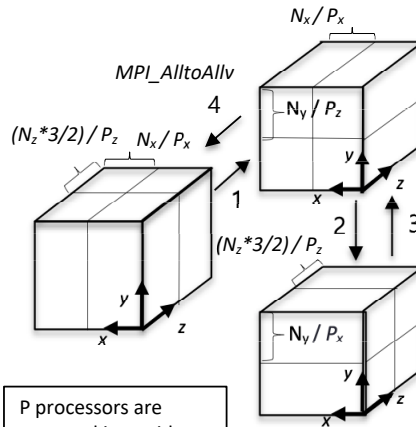
Number of communications per an iteration for matrix transpose (MPI\_AlltoAllv): 2

Buffer size per process (per comm.):  $N_x * N_y * (N_z * 3/2) / P^2$

Send/recv times per process (per comm.): P-1

Number of processes: P, where P should be less than  $N_y$  and  $(N_z * 3/2)$

1D decomposition



P processors are arranged in a grid so that  $P = P_x * P_z$  ( $4=2*2$ )

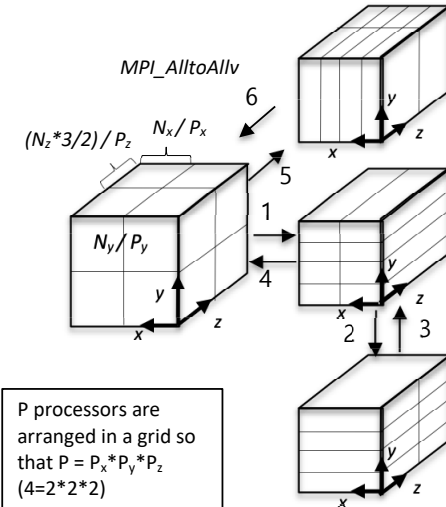
Number of communications per an iteration for transpose (MPI\_Alltoallv): 4

Buffer size per process (per comm.):  $(N_x * N_y * (N_z * 3/2)) / (P_x * P_z * P_z)$

Send/recv times per process (per comm.):  $P_x - 1$  or  $P_z - 1$

Number of processes:  $P = P_x * P_z$ , where  $P_x$  and  $P_z$  should be less than  $N_x$  and  $(N_z * 3/2)$ , respectively

2D decomposition



P processors are arranged in a grid so that  $P = P_x * P_y * P_z$  ( $4=2*2*2$ )

Number of communications per an iteration for transpose (MPI\_Alltoallv): 6

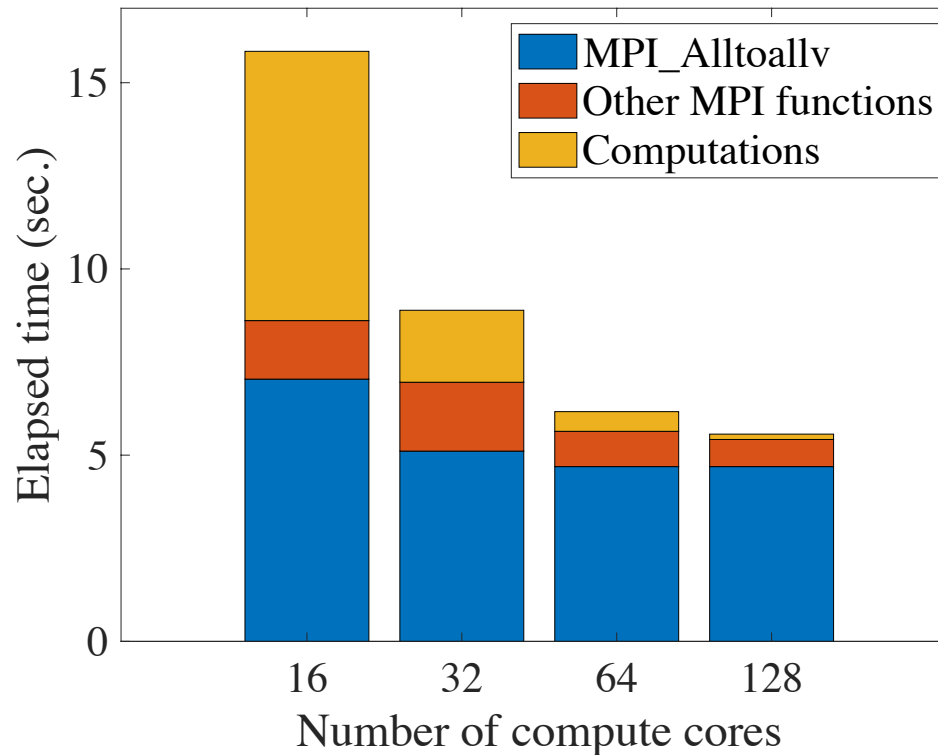
Buffer size per process (per comm.):  $(N_x * N_y * (N_z * 3/2)) / (P_x * P_y * P_z * P_z)$

Send/recv times per process (per comm.):  $P_x - 1$  or  $P_y - 1$  or  $P_z - 1$

Number of processes:  $P = P_x * P_z$ , where  $P_x$ ,  $P_y$  and  $P_z$  should be less than  $N_x$ ,  $N_y$  and  $(N_z * 3/2)$ , respectively

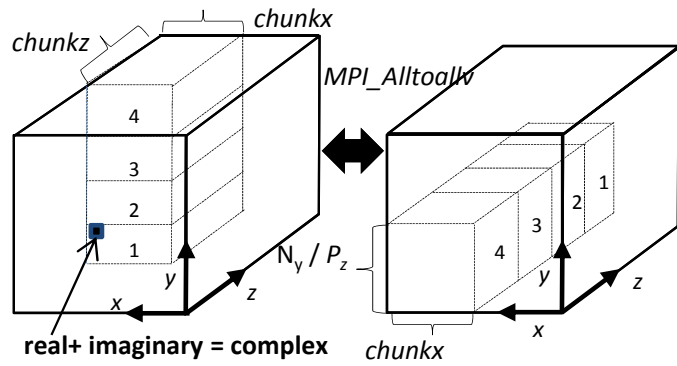
3D decomposition

Profiling results of 1D domain decomposition (small case: 128x129x128)



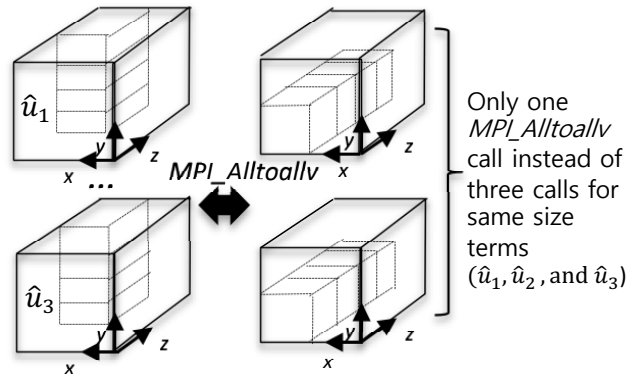
Computations indicates the scalability of computations except the MPI communications and other MPI functions include MPI Send, MPI Recv, MPI Allreduce, MPI Bcast, and MPI Barrier.

## Schematics of data transposition methods.



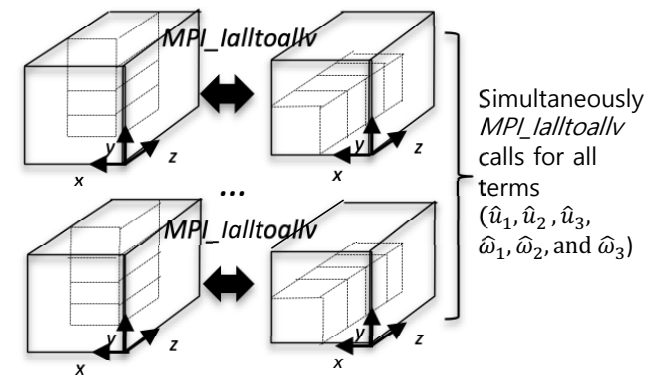
$MPI\_Alltoallv$  is called using the complex data structure, while original communication had been respectively called for the real and the imaginary parts of the complex number.

(a) Re-organization of the data structure(base line)



Only one  $MPI\_Alltoallv$  call instead of three calls for same size terms ( $\hat{u}_1, \hat{u}_2$ , and  $\hat{u}_3$ )  
Each block of same size terms (e.g.  $\hat{u}_1, \hat{u}_2$ , and  $\hat{u}_3$ ) is packed and communicated with  $MPI\_Alltoallv$ .

b) Use of terms packing



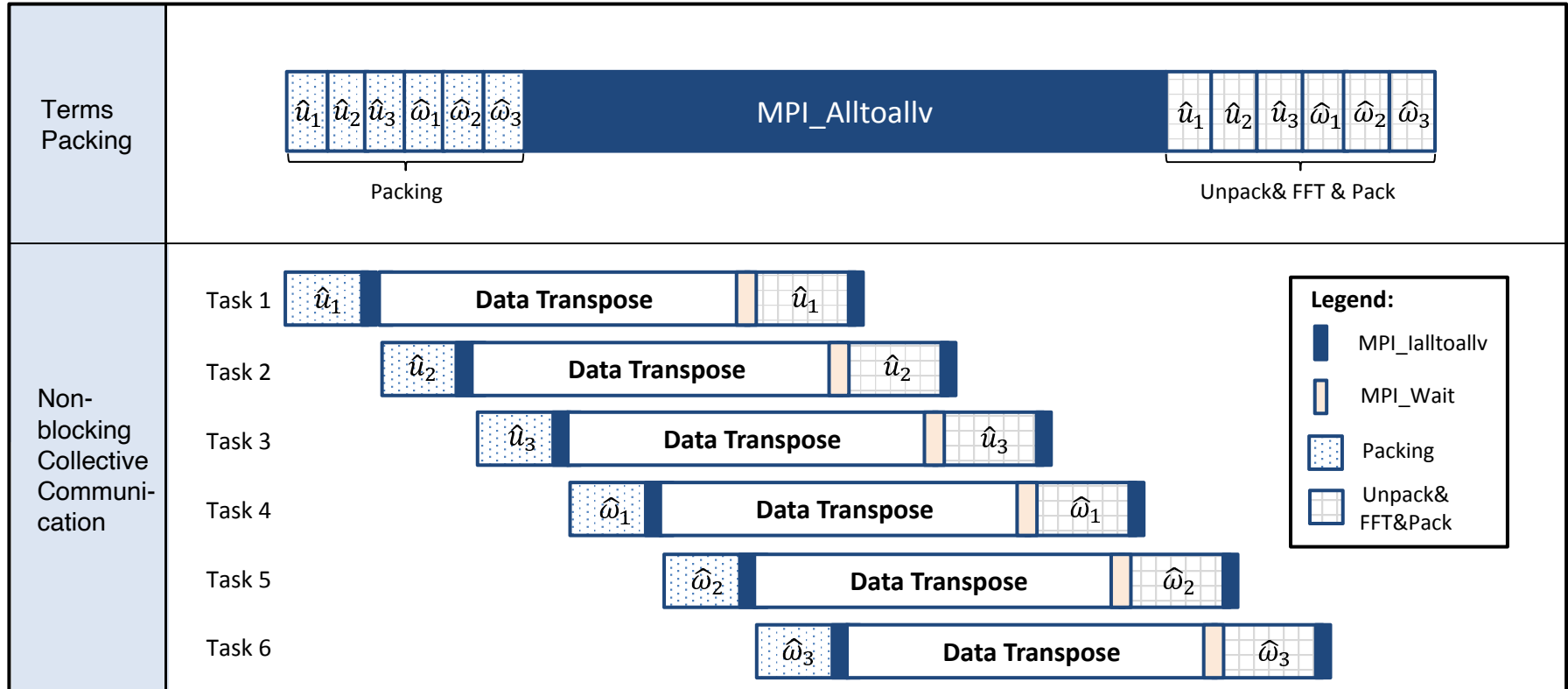
Simultaneously  $MPI\_lalltoallv$  calls for all terms ( $\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{\omega}_1, \hat{\omega}_2$ , and  $\hat{\omega}_3$ )  
Each term is overlapped between communication and computation using  $MPI\_lalltoallv$  of non-blocking collective communication operation.

(c) Use of non-blocking communication

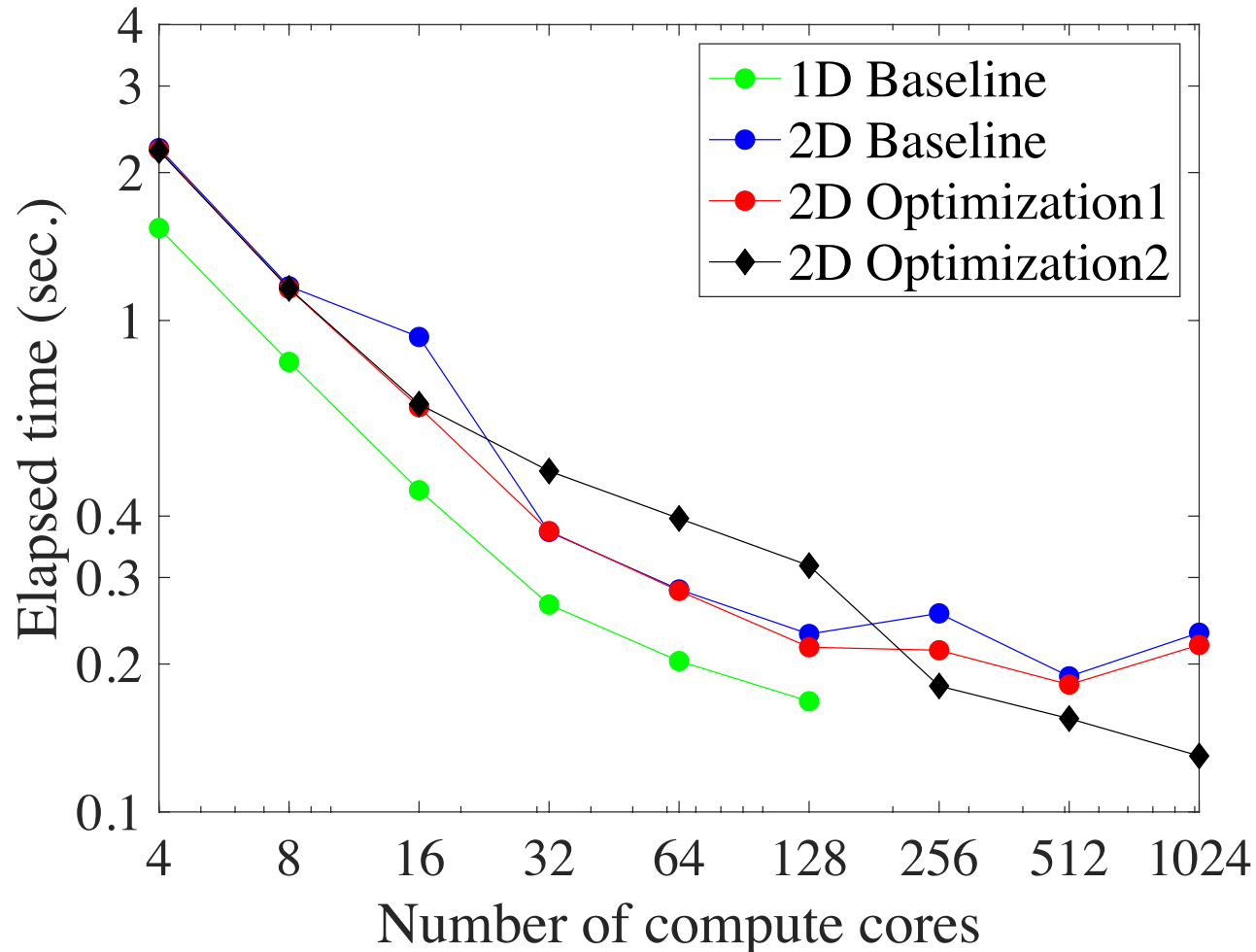
Note that  $P (= P_x \times P_z)$  processors are arranged in a grid with  $chunkx = (N_x \times 3/2)/P_x$  and  $chunkz = (N_z \times 3/2)/P_z$ , where  $N_x, N_y$  and  $N_z$  are the numbers of grid points along the x, y, and z-axes, respectively.



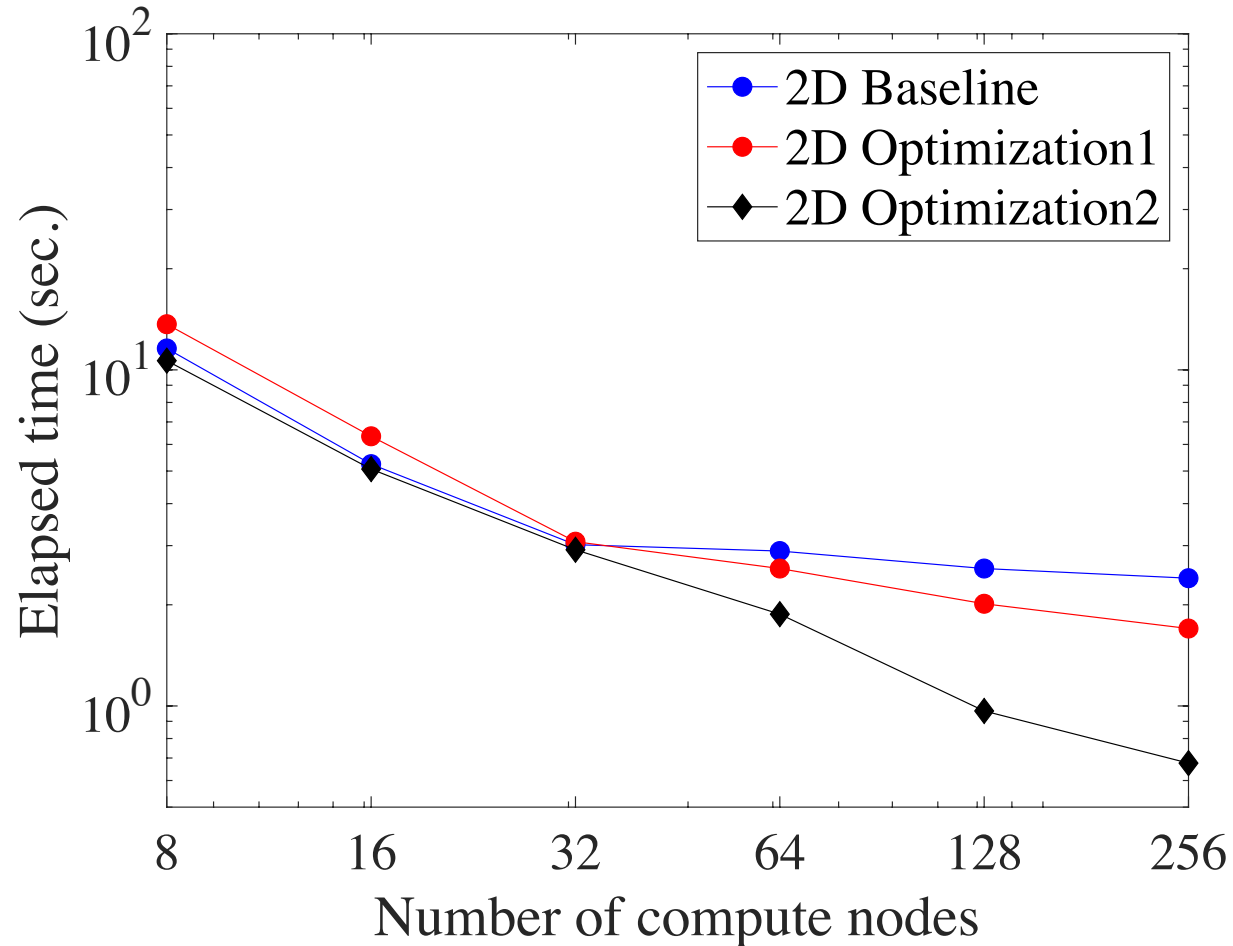
Differences of the transposition methods between terms packing and non-blocking collective communication.



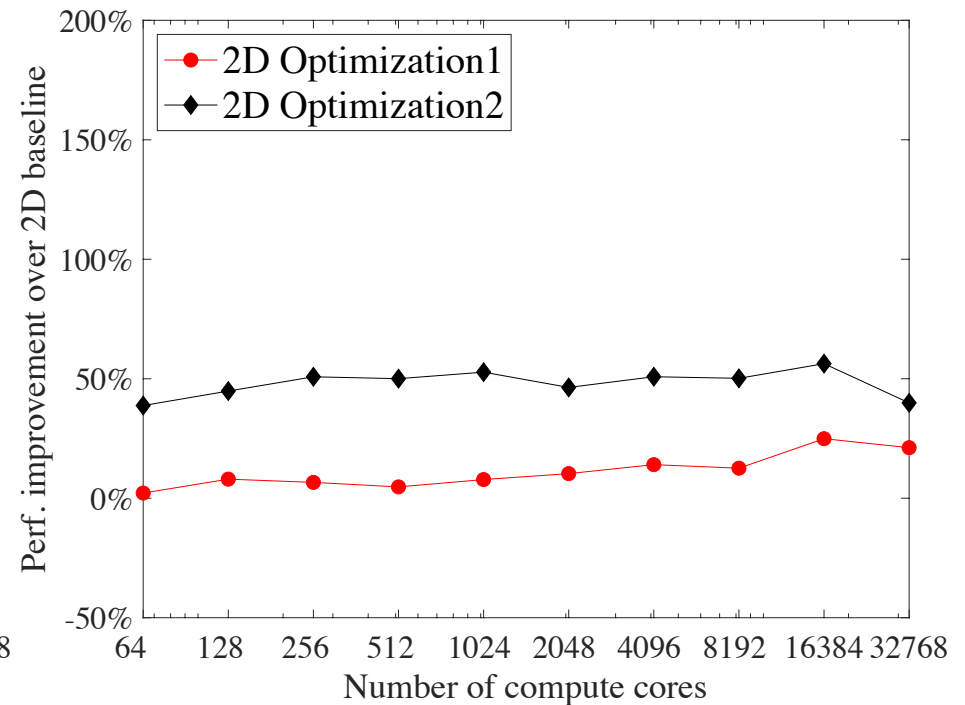
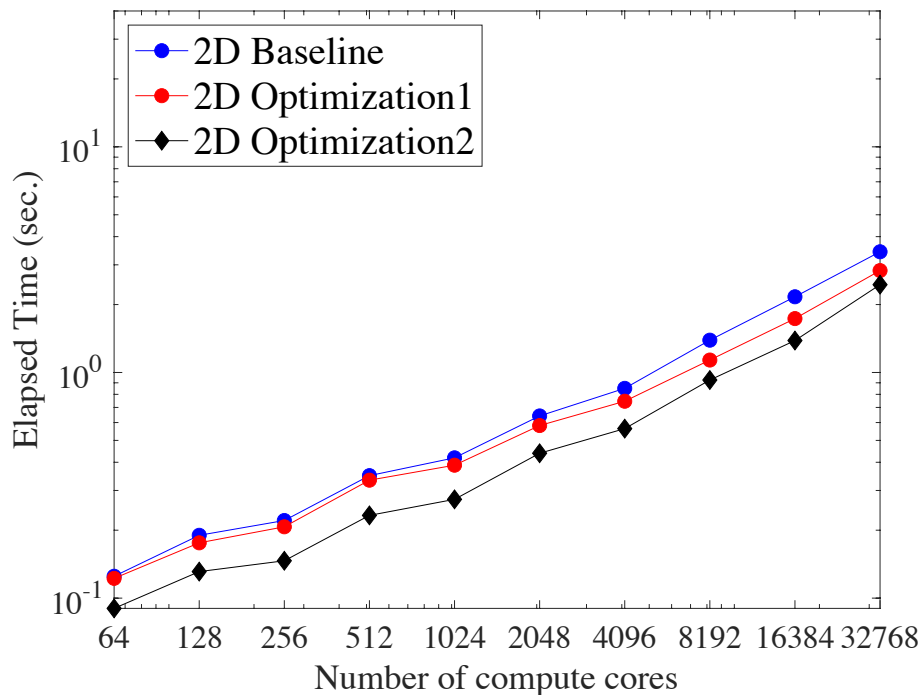
Performance comparison using different transposition algorithms: strong scalability for the small case: 128x129x128 (ReT=180)



Performance comparison using different transposition algorithms: strong scalability for the big case: 1536x257x1536 (ReT=550)



Performance comparison using different transposition algorithms: weak scalability for the  $64 \times 65 \times 64$  grid points per each compute node.



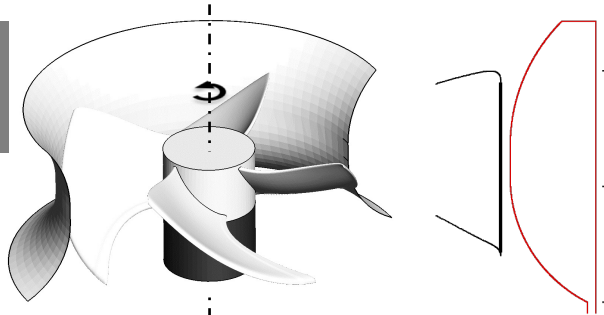
LES를 이용한 축류팬 주위  
유동 해석 코드 개발 @  
4호기 Tachyon



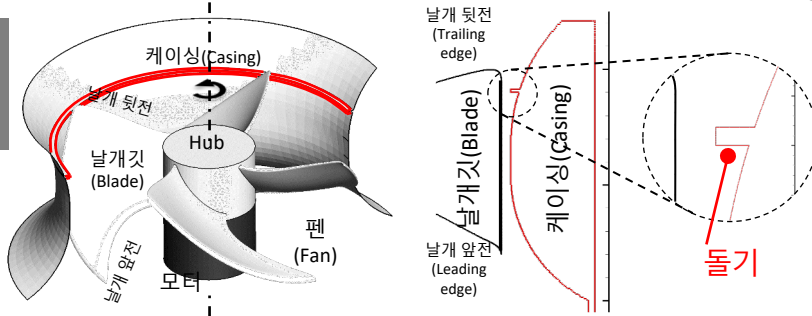
권오경  
슈퍼컴퓨팅응용센터  
국가슈퍼컴퓨팅본부



기존 제품



돌기가 적용된 제품



- 팬이 지나가는 공기 유동을 예측하여 효율을 떨어뜨리고 소음을 발생시키는 주범인 소용돌이를 슈퍼컴퓨팅 실험을 통해 확인
- 소용돌이를 발생시키지 않고 바람이 잘 흐르도록 돌기를 달아 유동을 제어하게 함으로써 시스템 에어컨의 풍량 증가, 소음 감소를 실현

- 에어컨 실외기 팬 주위의 바람을 모사
  - LG전자 시스템에어콘 팬에 돌기를 적용함으로 풍량을 증가하고 소음을 줄이게 함
  - 6,000개 이상 CPU를 동시에 사용할 수 있는 코드를 개발

전자신문

2018년 3월 5일 월요일 022면 전국

## 시뮬레이션 SW로 고효율·저소음 에어컨 실현

**KISTI·서울대, 슈퍼컴퓨팅 활용  
공기 유동 정밀예측...팬 성능 향상**

국내 연구진이 슈퍼컴퓨팅 시뮬레이션 소프트웨어(SW)를 활용해 고효율·저소음 시스템 에어컨 제품 실현에 성공했다. 슈퍼컴퓨팅을 활용한 우수 산업 성과 창출 사례로 평가된다.

한국과학기술정보연구원(KISTI·원장 최희윤)은 최해천 서울대 기계항공공학부 교수팀과 함께 에어컨 실외기 팬 주위의 바람을 모사하는 거대 규모의 슈퍼컴퓨팅 시뮬레이션 SW를 최근 개발했다.

공동 연구팀은 시뮬레이션 SW로 팬 주위의 바람 현상을 해석, LG전자의 시스템에어컨에 적용했다. 시뮬레이션 SW는 KISTI 슈퍼컴퓨터 4호기의 중앙처리장치(CPU)를 최대 6000개 이상 동시 사용했다. 팬 주위의 공기 유동 상태를 기존 대비 8배 이상 정밀하게 예측, 에너지 효율을 떨어뜨리고 소음을 발생시키는 소용돌이를 확인했다. 에너지 효율과 소음은 팬의 성능을 나타내는 주요 지표다.

또 시뮬레이션 결과를 토대로 소용돌이 없이 바람이 잘 흐르도록 유로에 돌기를 달아 에어컨의 풍량 증가, 소음 감소를 실현했다.

연구팀은 KISTI의 슈퍼컴퓨팅으로 연구 및

제품 적용 성과를 거뒀다고 설명했다. 실험을 이용하는 기존 방법으로는 팬 회전 시 유동을 정확하게 예측하는 것이 어렵다. KISTI는 최적 병렬화 기술 및 계산 자원을 이용자에게 지원하고 있다. 최적병렬화는 수천 개의 CPU가 동시에 업무를 수행하도록 코드를 개발, 어려운 문제를 해결하는 기술이다.

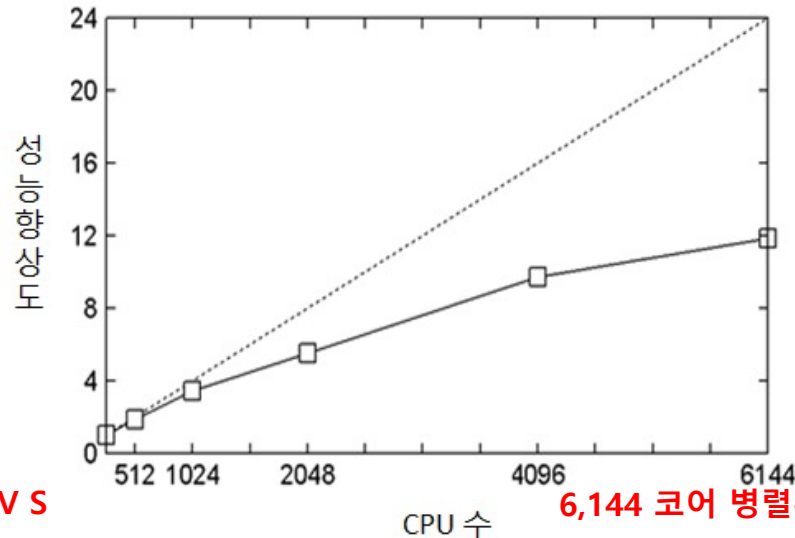
황순욱 KISTI 슈퍼컴퓨팅본부장은 “슈퍼컴퓨팅시뮬레이션 SW로 기업 제품 개발 주기 감소, 비용 절감에 대전기가 마련될 것”이라면서 “성능이 기존 대비 70배 향상될 슈퍼컴퓨터 5호기도 이를 뒷받침할 것”이라고 말했다.

대전=김영준기자 kyj85@etnews.com

### SW 개발 관련 주요 일간지 보도 (18년 3월 5일)



시스템에어콘 MULTI V S



6,144 코어 병렬확장

## • 축류 팬

- 팬이 회전하면서 다량의 공기를 팬의 앞쪽 혹은 뒤쪽으로 불어주는 유체 기계
    - 예) 선풍기, 전자제품의 열을 식혀주는 팬, 건물 내부 공기 환기 팬
  - 팬 소음 해석을 위한 수치해석 프로그램
    - 실험으로는 팬 주위 3차원 유동의 비정상 특성이나 소음의 전파 특성을 분석하기 어려움
    - 실험을 대신하여 수치해석 프로그램을 통한 연구가 활발함
- ➔ 본 연구는 팬 주위의 소음 해석을 위한 고병렬 프로그램의 최적화 기법

Outdoor unit of  
an air-conditioner



www.proidea.ro

IT fan



www.blacknoise.com

Industrial Ventilation



www.lorencook.com

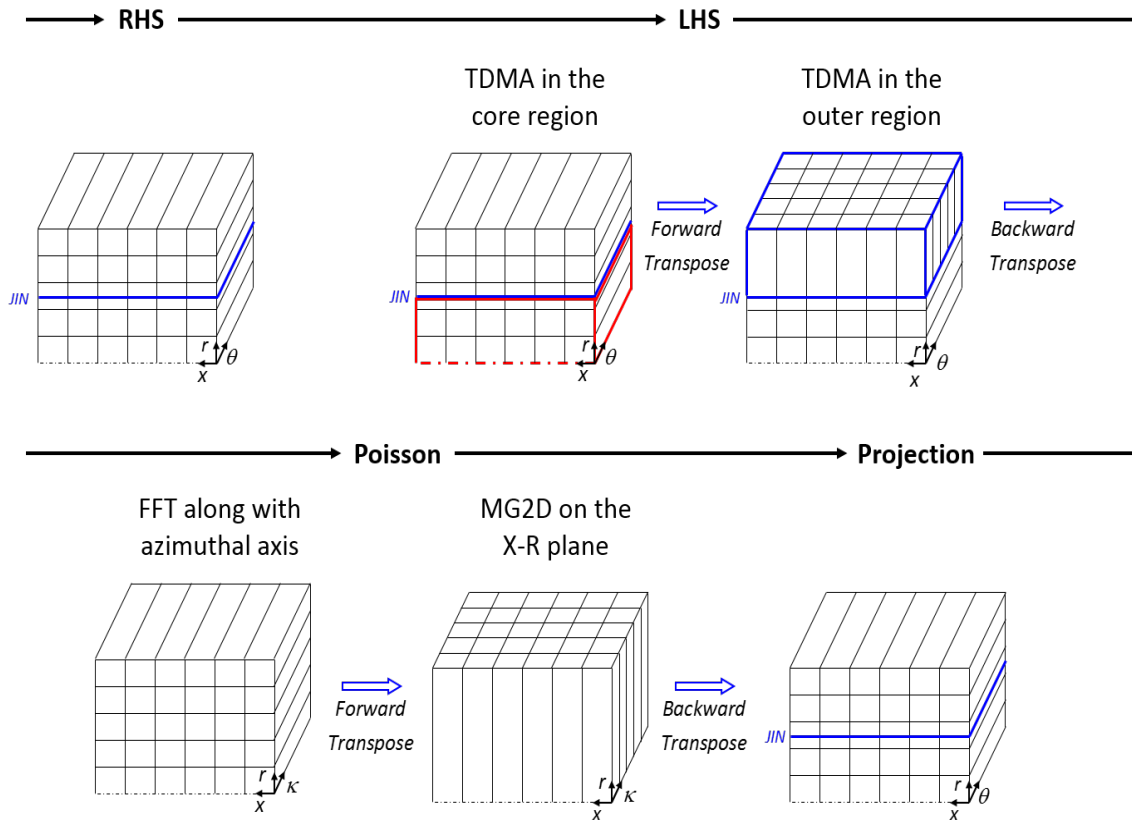


## ▶ 축류 팬 해석 프로그램

- 비관성-원통형 좌표계에서 Navier-Stokes 방정식과 연속 방정식을 계산하여 유동을 해석한 후, 유동 정보를 바탕으로 소음을 별도로 해석
- 축류 팬 주위 유동 해석에서 가장 중요한 유동 현상으로는 날개와 외부 관(Duct) 사이의 좁은 간극에서 발생하는 익단 누설 유동(tip leakage flow)이 있으며, 이를 해석하기 위해 최소로 요구하는 격자 수준은 약 4억 4천만 개.
- 소음을 해석하기 위해서, 목표로 하는 격자 수준은 약 8억 개이며, 소음해석을 위한 포인트 개수는 약 2만 개
- 이를 계산하기 위해서는 수백 기가 바이트 이상의 메모리와 수천 CPU 코어를 가진 슈퍼컴퓨터가 필요
  - ➔ 해당 계산을 분산메모리 환경에서 계산하기 위해 MPI 병렬화로 구현
  - ➔ 팬 소음 해석 시 메모리 사용량 및 각 계산 노드간 통신 횟수의 급격한 증가로 인해 통신이 멈추는 현상이 발생
    - MPI 일방향통신(one-sided communication) 방법을 적용 멈춤 현상 해결
    - 병렬확장성을 확보하기 위해 메모리 사용량을 최적화 하고, 통신 횟수를 줄여 성능 향상시킴

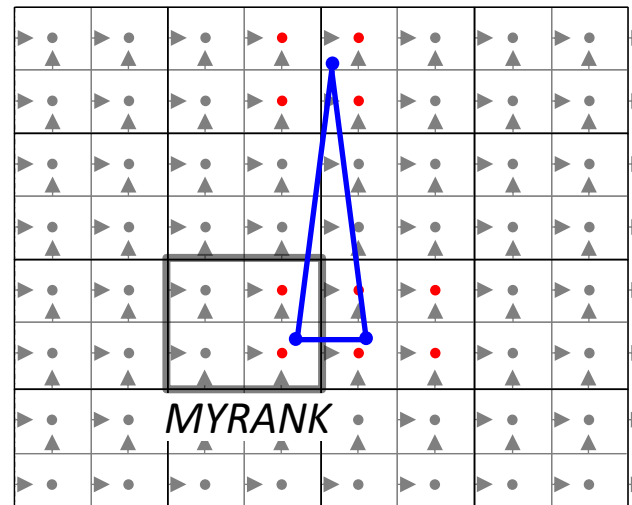
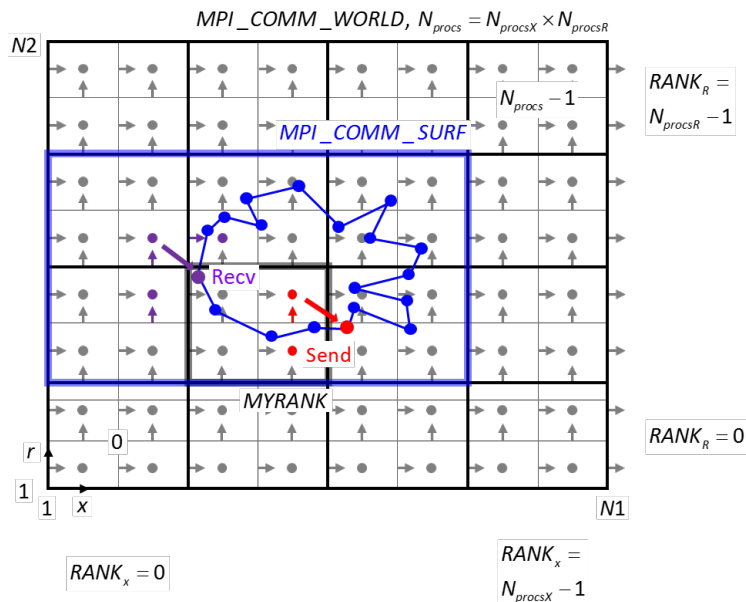
## ▶ 병렬 설계

- 분산메모리 환경의 슈퍼컴퓨터에서 병렬확장성을 확보할 수 있게 2차원 분할
  - 축류 팬 해석을 위해 원통형 좌표계를 사용하여 좌표계는  $x, r, \theta$



## ▶ 소음 해석을 위한 설계

- 소음 설계를 위한 별도의 Communicator 정의
  - 유동 계산은 MPI\_COMM\_WORLD, 소음 계산은 MPI\_COMM\_SURF에서 통신이 수행
- 소음 한 포인트에서 발생하는 소음을 계산하기 위해서는 해당 위치의 주위 속도 12개와 압력 8개 포인트 정보가 필요
- 계산 시 필요한 주위의 정보는 도메인이 2차원으로 분할되어 있기 때문에 MPI 통신을 통해 받아와야 함



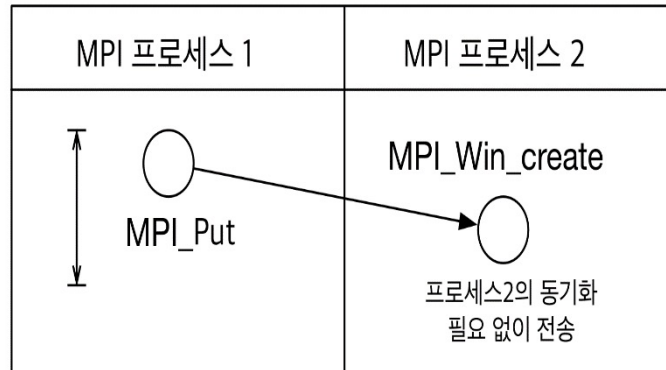


## ▶ 소음 해석시 문제

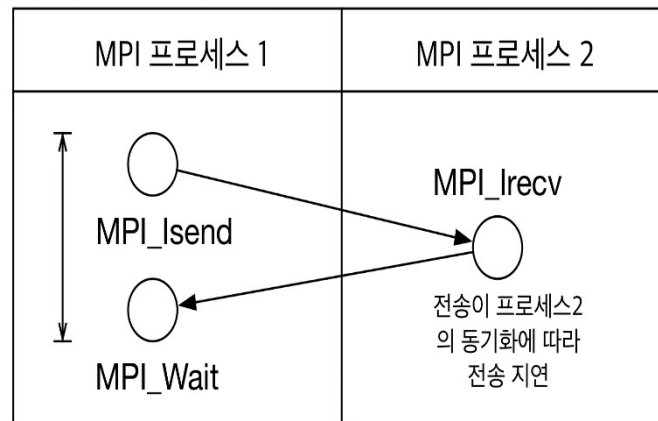
- 1회 통신시 데이터 버퍼 크기는 크지 않음: 480바이트
  - 하지만 MPI 프로세스별로 여러 포인트가 포함되어 있고, 주위의 여러 MPI 프로세스에서 포인트 정보를 가져와야 하므로 통신횟수 최대 4000번 이상일 수 있음
  - 특히 한 번에 수행하는 MPI 프로세스 수가 수천 개 이상일 경우에 한 번에 발생하는 통신 횟수가 백만 번 이상으로 발생하여 계산속도가 심각하게 느려짐
- ➔ KISTI 슈퍼컴퓨터 4호기 타키온2에서 실험 시 통신 멈춤 현상이 발생

전송 MPI 프로세스 번호	수신 MPI 프로세스 개수	통신 데이터 버퍼 크기	통신횟수
2	1	480바이트	10
4	1	480바이트	6
10	6	480바이트	2666
11	7	480바이트	1524
12	5	480바이트	936
18	4	480바이트	2699
19	7	480바이트	1647
20	5	480바이트	500
26	6	480바이트	4592
27	8	480바이트	2931
...	...	...	...

## ▶ 일방향통신을 통한 통신 멈춤 현상 해결



[ MPI 프로세스간 일방향통신(One-sided communication) 개념도 ]



[ MPI 프로세스간 양방향통신(Two-sided communication) 개념도 ]



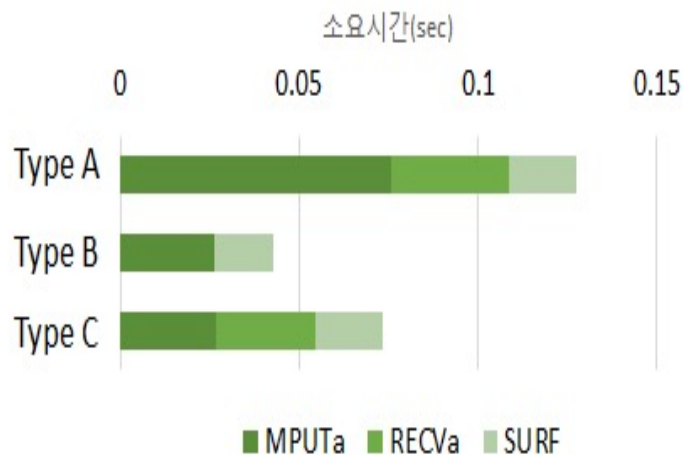


## ▶ 소음 통신 관련 최적화

- 통신 횟수 줄이기
  - 송신처에서 임시 버퍼를 할당해 개별 MPI 프로세스별로 전송하는 데이터를 모아서 전송하는 방식 사용
  - 해당 수신처의 MPI 프로세스 임시버퍼에서 메모리복사를 해야하는 오버헤드는 발생
- 메모리 사용량 줄이기
  - 축류 팬 표면을 구성하고 있는 삼각형 메시의 꼭지점에 대해서 미리 계산하고 결과 값만을 보내기 때문에 꼭지점에서 정의된 9개 정보만 통신하게 함

## ▶ 소음 통신 관련 최적화 결과

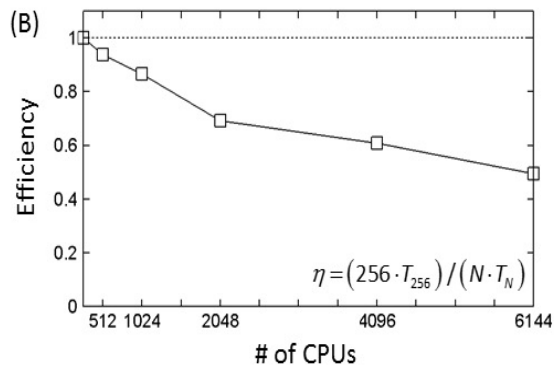
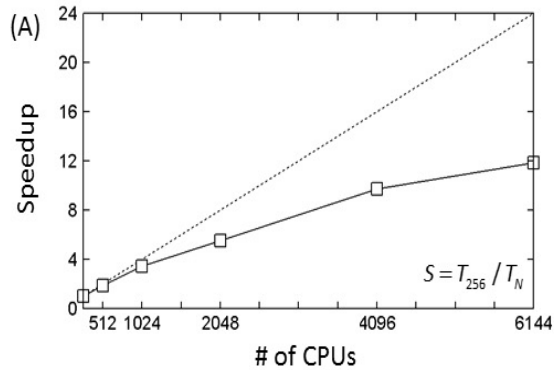
- 통신 횟수 감소를 통해서 1.74배, 메모리 사용량 최적화를 통해 최대 2.97배 향상
- Type A는 MPI 일방향통신만 적용하고 최적화를 적용하지 않은 버전
- Type B는 통신횟수 및 메모리 사용량 줄이는 방법 모두를 적용한 버전
- Type C는 통신 횟수를 줄이는 방법만 적용한 버전



MPUTa: MPI 통신시간, RECVa: 메모리 복사 시간, SURF: 표면 계산 시간,  
4천 6백만 격자, 848 소음포인트  
x축: 소요시간, y축: 최적화방법

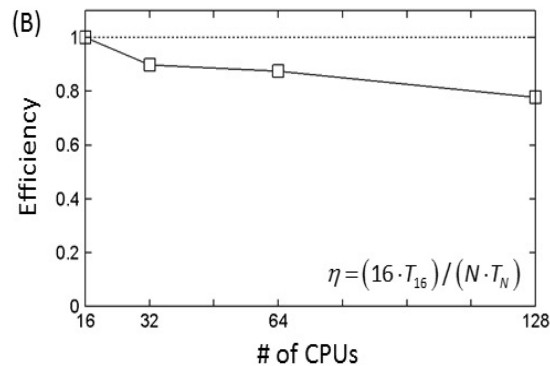
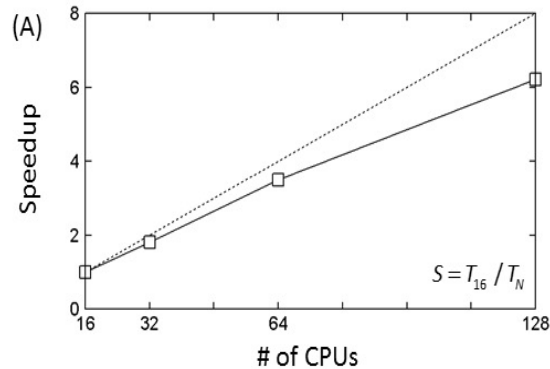
## ▶ 유동해석 코드의 병렬확장성 (16억개 격자)

- (A) Tachyon2에서 6144코어까지의 병렬확장성
- (B) 병렬효율은 최대 약 50%. 유동 계산 시 행렬 전치를 위해 집합통신이 사용되어 코어가 증가함에 따라 병목현상이 발생




## ▶ 유체/소음 코드의 병렬확장성 (4,600만 개의 격자와 67,000개의 소음원 포인트)

- (A) Tachyon2에서 128코어까지의 병렬확장성
- (B) 병렬효율은 128코어에서 약 80%의 효율. 격자수를 8억 개 이상으로 증가시켜도 비슷한 병렬효율성을 보여줄 것으로 예상





- 축류 팬(axial fan)은 선풍기와 같은 팬이 회전하면서 다량의 공기를 불어주는 유체 기계
- 수억 개의 격자와 수만 포인트의 소음을 해석하기 위해서 2차원 분할 방법을 사용하여 Tachyon2에서 최대 6144코어까지 수행가능한 모델 개발
- 수만 포인트의 소음 해석시 각 MPI 프로세스간 통신이 많이 발생하여 멈춤 현상이 발생. 이를 극복하기 위해 MPI 일방향통신을 적용하였고, 통신 최적화를 통해 1.74배, 메모리 최적화 방법을 통해 최대 2.97배 향상
- 이론적 접근이나 실험만으로는 축류 팬에 의해서 유도되는 복잡한 3차원 난류 유동을 정확하게 예측/측정할 수 있는 방법이 매우 제한적인 상황에서, 슈퍼컴퓨터를 이용한 수치 해석을 통해 축류 팬 주위 유동 구조 분석을 제시
- 개발한 코드는 축류 팬뿐만 아니라 다양한 유동 현상에 대해서 범용적으로 활용할 수 있어 난류 유동의 연구 경쟁력을 높일 수 있음
- 향후 5호기 슈퍼컴퓨터에서 목표로 하는 큰 격자수와 소음원 개수를 가진 소음 계산의 병렬확장성을 실험할 예정



LES를 이용한 유체-고체 연성 해석을 통한  
 $Re \geq 10,000$  난류 영역에서의  
타코마 다리 붕괴 시뮬레이션 @ 5호기



권오경  
슈퍼컴퓨팅응용센터  
국가슈퍼컴퓨팅본부

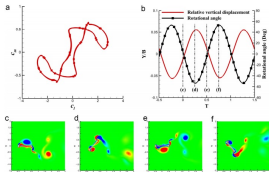
## 다리 주위 난류 유동 해석 및 핵심 유동 현상 규명

- ✓ 고정밀 수치해석 코드 개발 및 유동 구조 분석

### 선행연구 사례 및 한계



Akashikaikyō 교의 내풍실험, 일본



Lee et al., 2016

풍동실험/CFD를 이용한  
선행연구 사례

2차원 RANS 기법을  
적용한 수치해석 연구

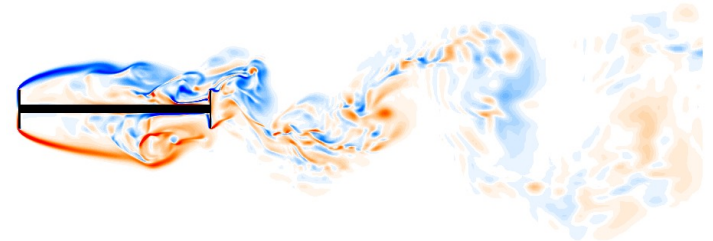


유동장 예측 정확도가  
현저히 낮음

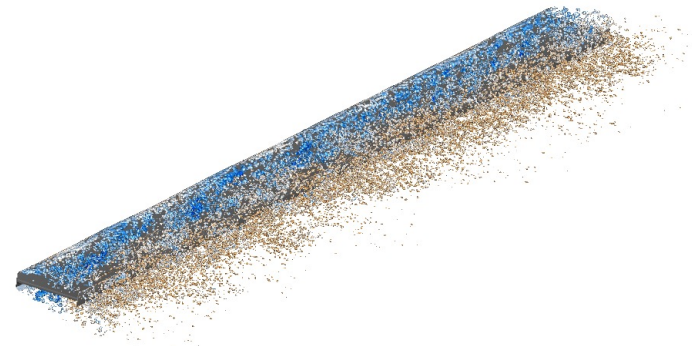
기존의 선행연구에 사용된 수치 기법으로는  
고정밀 유동장 해석 및 예측 불가능

복잡한 3차원 유동구조를 해석하고  
교량의 응답 특성을 정확히 예측할 수 있는  
고정밀 CFD 기술 개발 필수적

- ✓ 다리 주위의 유동은 복잡한 3차원 비정상 난류 유동에 해당



- ✓ 다리의 경간 방향으로의 3차원 유동

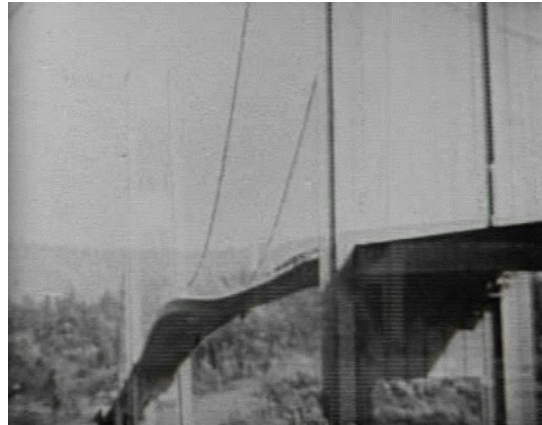


다리 주위 유동에 대한  
고정밀 수치해석

(큰 에디 모사,  
Large eddy simulation; LES) 필요

## 왜 타코마 다리인가?

- 타코마 다리의 붕괴는 1940년에 발생한 대형 교량 붕괴사고로, 다른 붕괴 사고와 달리 다리의 공탄성적(aeroelastic) 특성으로 인한 붕괴사고 (대학교 유체역학 교재에도 소개 되는 널리 알려져 있는 대표적인 사고)



### Vertical vibration

- 진동수 = 0.6-0.63 Hz
- 파장 =  $(2/10)L_T \sim (2/9)L_T$   
( $L_T$  = mid-span length)
- 진폭 = 0.91 m

### Torsional vibration

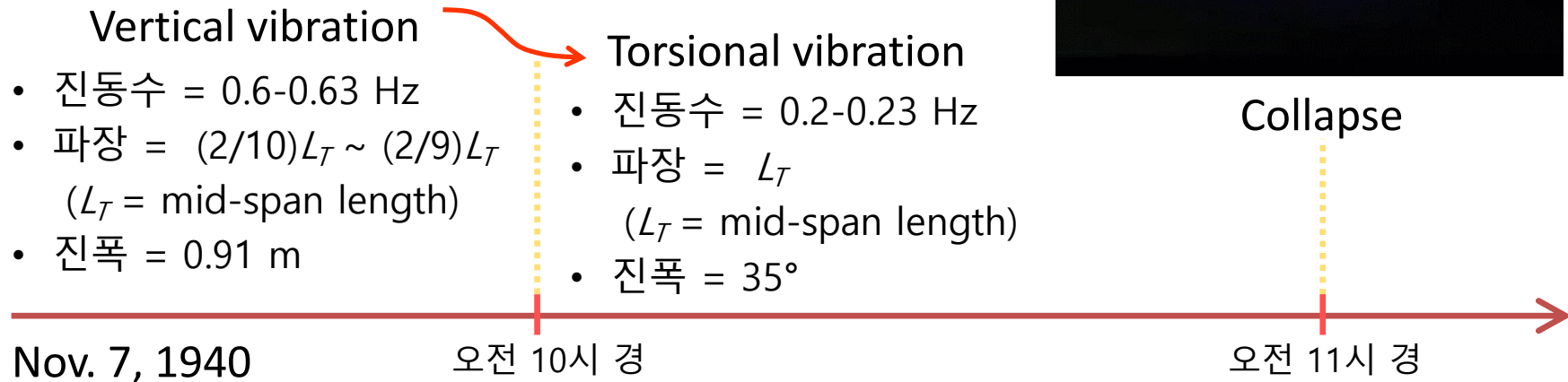
- 진동수 = 0.2-0.23 Hz
- 파장 =  $L_T$   
( $L_T$  = mid-span length)
- 진폭 = 35°

Nov. 7, 1940

오전 10시 경

오전 11시 경

Collapse





## 유동에 의한 교량의 진동 분석

### 유체-고체 상호작용

- 바람과 구조물의 상호작용에 의해 발생하는 거동을 분석하기 위해 유체-고체 상호작용(fluid-structure interaction, FSI)에 기반한 연구 수행
- 기존의 교량 FSI 연구는 2차원으로 이루어진 경우가 많은데, 이는 공간(span)방향으로 발달하는 3차원적 진동 모드를 해석하기 불가능

### 난류 유동을 위한 큰 에디 모사

- 기존의 교량 FSI 연구는 계산 비용이 저렴한 RANS 방정식을 이용하여 난류를 모델링한 경우가 많음
- 이는 후류에 의해 교량에 작용하는 힘과 그로 인해 발생하는 구조의 거동을 분석하기에 적합하지 않음
- 강한 바람으로 초래되는 난류 유동의 비정상적 특성을 추적하기에 용이한 큰 에디 모사 (large eddy simulation, LES) 를 사용

- LES를 이용한 고정밀 고해상도 FSI solver 개발
- 타코마 다리의 전 붕괴 과정을 최초로 수치해석으로 재연

## 유체

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial \bar{u}_i \bar{u}_j}{\partial x_j} = -\frac{\partial \bar{p}}{\partial x_i} + \frac{1}{\text{Re}} \frac{\partial^2 \bar{u}_i}{\partial x_j \partial x_j} - \frac{\partial \tau_{ij}}{\partial x_j} + f_i$$

$$\frac{\partial \bar{u}_j}{\partial x_j} - q = 0$$

$(\bar{\quad})$  : spatial filtering

$x_i$  : Cartesian coordinates ( $x, y, z$ )

$u_i$  : non-dimensional velocity ( $u, v, w$ )

$p$  : non-dimensional pressure

$\tau_{ij}$  : subgrid-scale stress tensor

$f_i$  : momentum forcing

$q$  : mass source/sink

- 동적 광역 모델 (Dynamic global model) (Park *et al.*, 2006; Lee *et al.*, 2010)

- 가상 경계 방법 (immersed boundary method) (Kim, Kim & Choi, 2001)

## 고체

$$(M + 2m\xi) Y_{c_{tt}}$$

$$= -EI Y_{c_{zzzz}} + H_0 \left( \frac{2Y_{c_z}^2}{\xi^2} + 3 \frac{s'(Y_{c_z}^2 + l^2 \theta_{c_z}^2)}{\xi^4} \right)_z$$

$$- \frac{AE}{L_c} \left[ \int_0^{L_z} \frac{Y_{c_z}^2 + l^2 \theta_{c_z}^2}{\xi^3} \right] \frac{s''}{\xi^3} - \frac{2AE}{L_c} \left[ \int_0^{L_z} \frac{s'' Y_c}{\xi^3} \right] \left( \frac{s'}{\xi} - \frac{Y_{c_z}}{\xi^3} \right)_z$$

$$+ \frac{2AE l^2}{L_c} \left[ \int_0^{L_z} \frac{s'' \theta_c}{\xi^3} \right] \left( \frac{\theta_{c_z}}{\xi^3} \right)_z + \rho_f U^2 h \frac{C_L}{2}$$

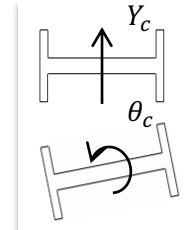
$$\left( \frac{M}{3} + 2m\xi \right) \theta_{c_{tt}}$$

$$= \frac{GK}{l^2} \theta_{c_{zz}} + 2H_0 \left( \frac{\theta_{c_z}}{\xi^2} + 3 \frac{s' Y_{c_z} \theta_{c_z}}{\xi^4} \right)_z$$

$$- \frac{2AE}{L_c} \left[ \int_0^{L_z} \frac{Y_{c_z} \theta_{c_z}}{\xi^3} \right] \frac{s''}{\xi^3} - \frac{2AE}{L_c} \left[ \int_0^{L_z} \frac{s'' \theta_c}{\xi^3} \right] \left( \frac{s'}{\xi} - \frac{Y_{c_z}}{\xi^3} \right)_z$$

$$+ \frac{2AE}{L_c} \left[ \int_0^{L_z} \frac{s'' Y_c}{\xi^3} \right] \left( \frac{\theta_{c_z}}{\xi^3} \right)_z + \rho_f U^2 h^2 \frac{C_M}{2l^2}$$

현수교 비선형 모델  
(Arioli & Gazzola, 2017)

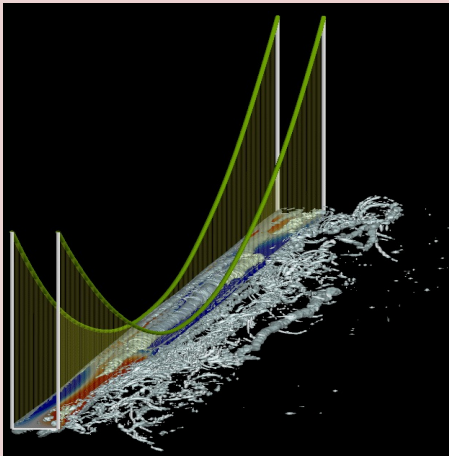


유체-구조 상호작용을 위한 weak coupling  
(Kim & Choi, 2018)

## 기존 계산 결과

Re=300

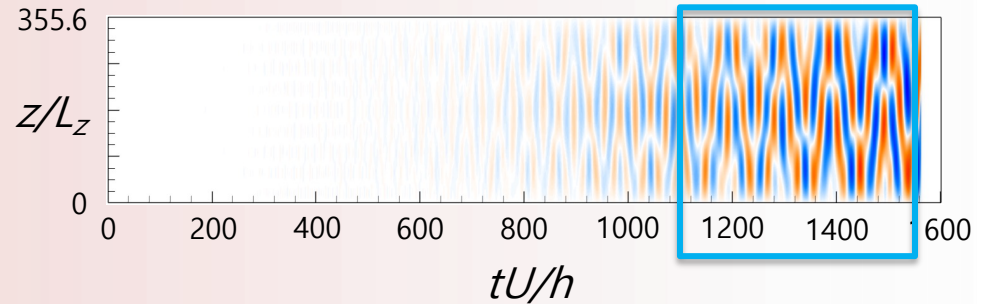
(약 4억개 격자)



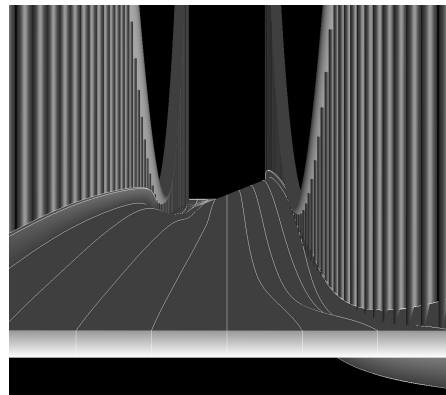
- Reynolds 수에 따른 진동수는 동일
- 그러나 비틀림 진동의 파장이 달라짐
- 작은 스케일 유동에서 차이가 남
- 난류 영역의 시뮬레이션이 필요

Torsional vibration  
 $\lambda_z = 2/3 L_z$  &  $f = 0.2$  Hz

$\theta_c$  -50° 0 50°

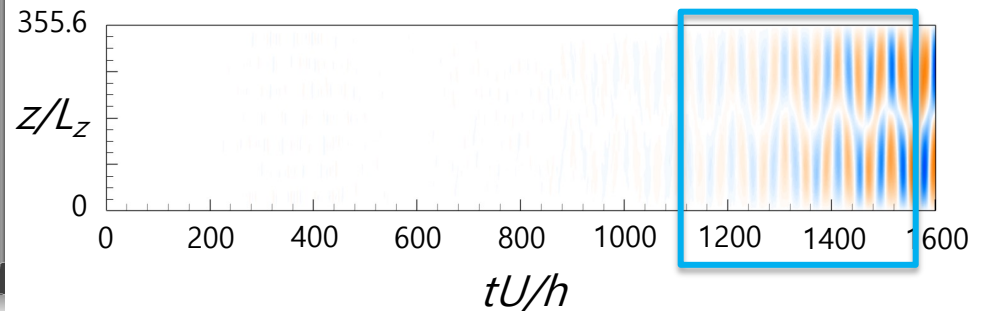


Re=1,000



Torsional vibration  
 $\lambda_z = L_z$  &  $f = 0.19$  Hz

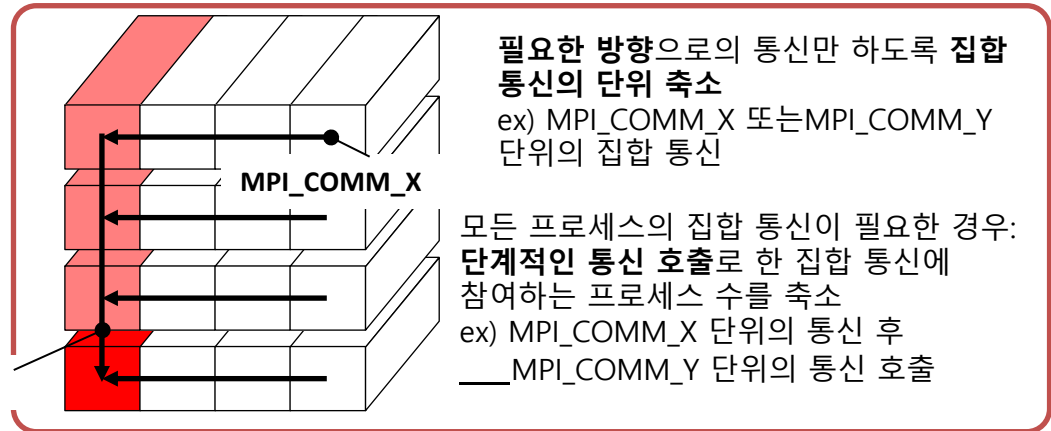
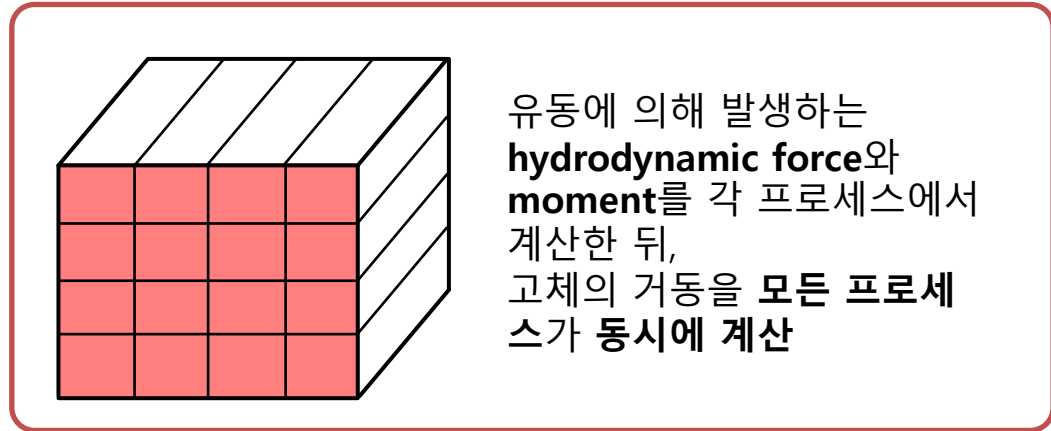
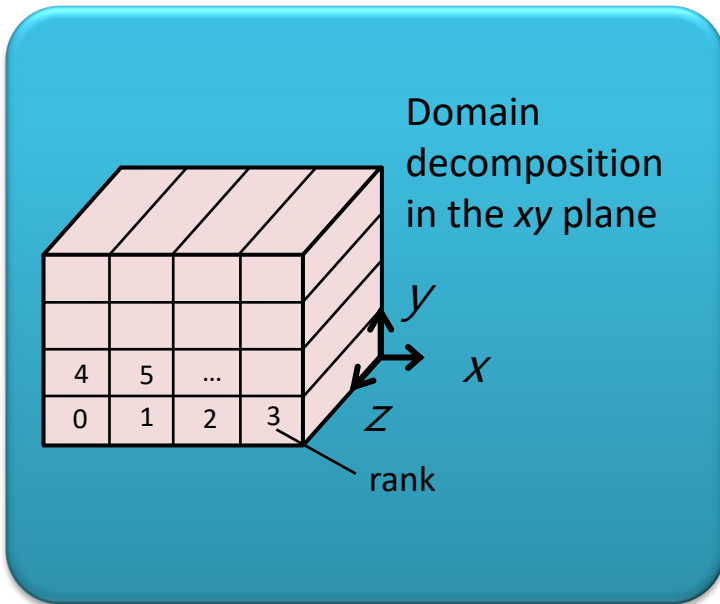
$\theta_c$  -40° 0 40°



→ Higher Reynolds number!!!

## MPI (Message-passing interface)를 이용한 병렬화

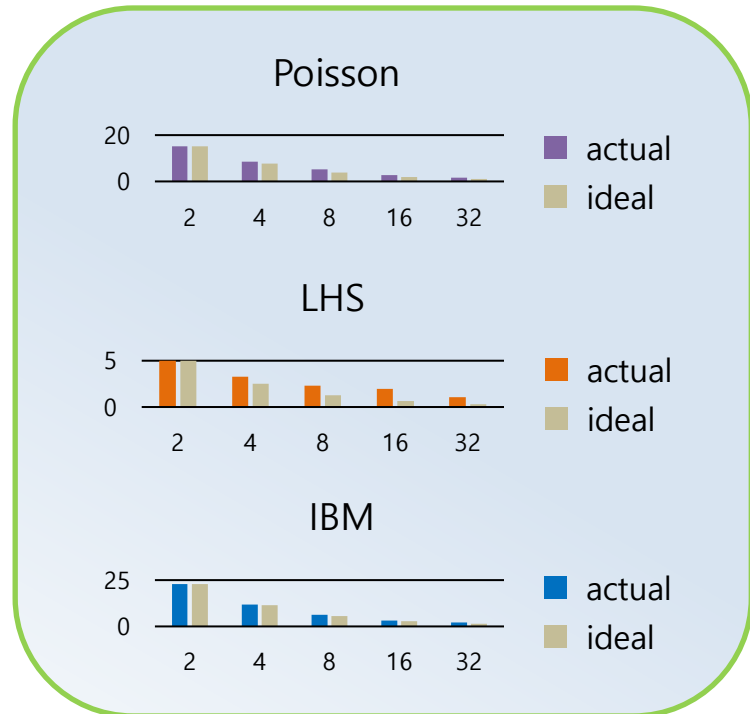
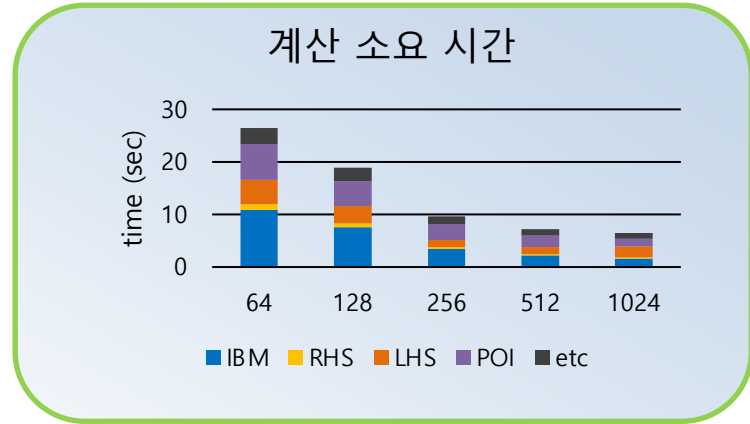
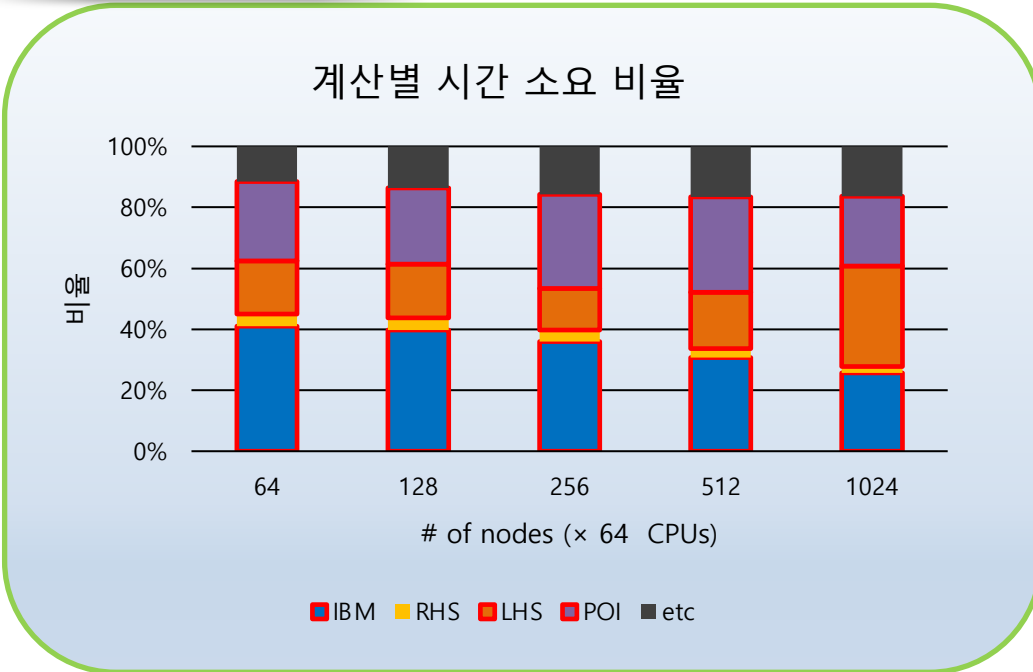
- 난류 영역의 높은 Reynolds 수 ( $Re = 10,000$ ) 계산을 위하여 80억 격자 예상
- 각 프로세스(process)에서는 x, y 방향으로 분할된 영역만을 담당하는 분산 메모리 환경 구현



MPI\_COMM\_Y

## 기존 병렬 성능

(4,097×2,049×512 = 4.3억 격자 사용)



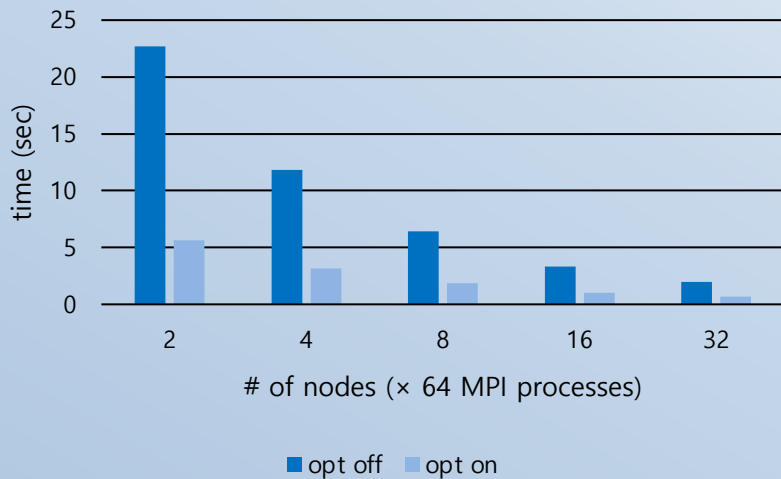
**Poisson solver** 의 메모리 호출 횟수가 많았음.

**LHS 계산 (TDMA)** 중 행렬의 전치 과정을 위한 MPI 통신 횟수가 많았음.

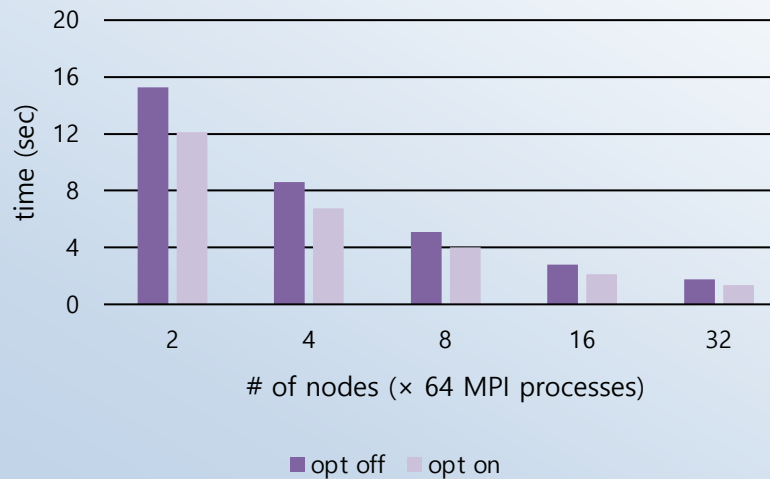
**IBM** 내에서 함수 호출 횟수가 많았음.

## 최적화된 병렬 성능

IBM



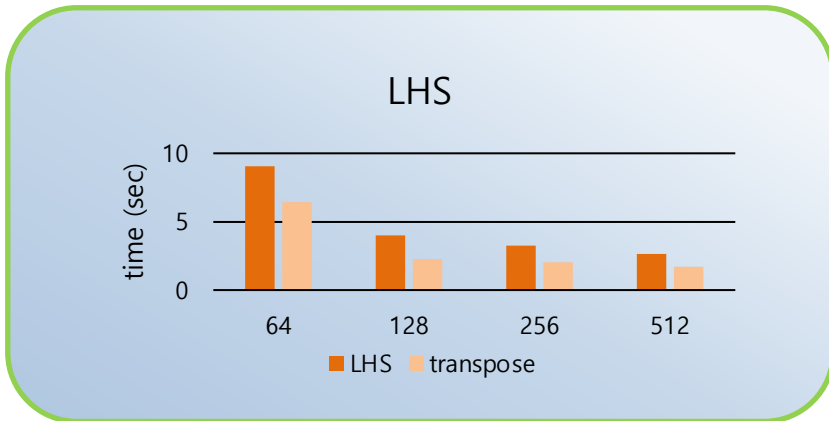
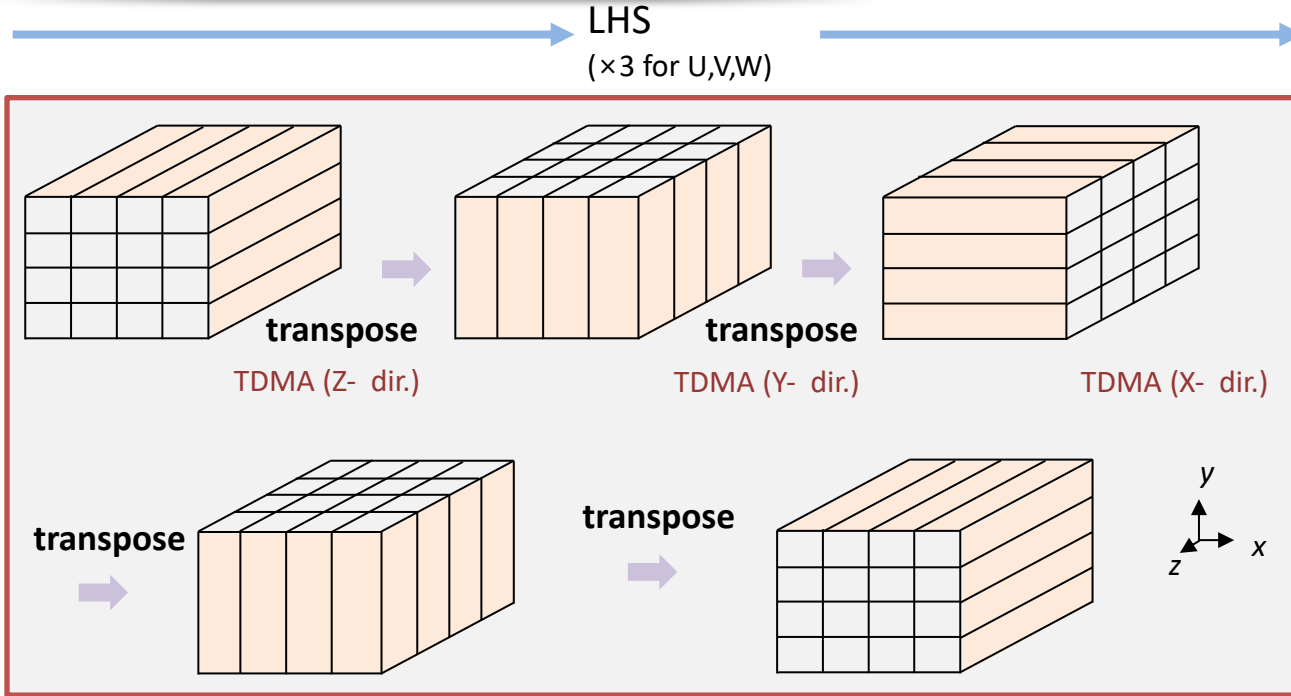
Poisson



**IBM** 내에서 적절한 변수 사용으로 함수 호출 횟수를 줄여 계산 시간을 절감함.

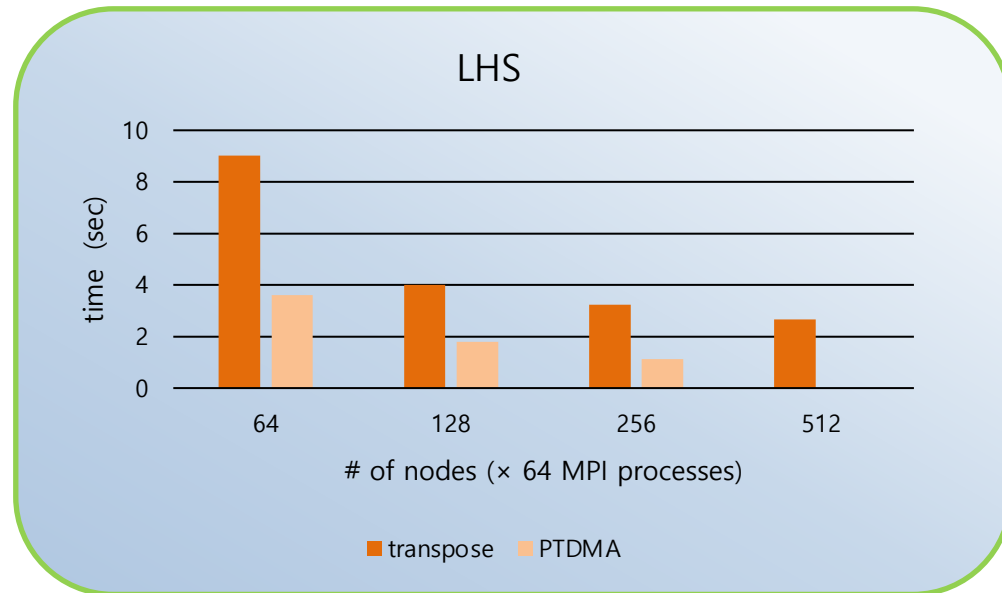
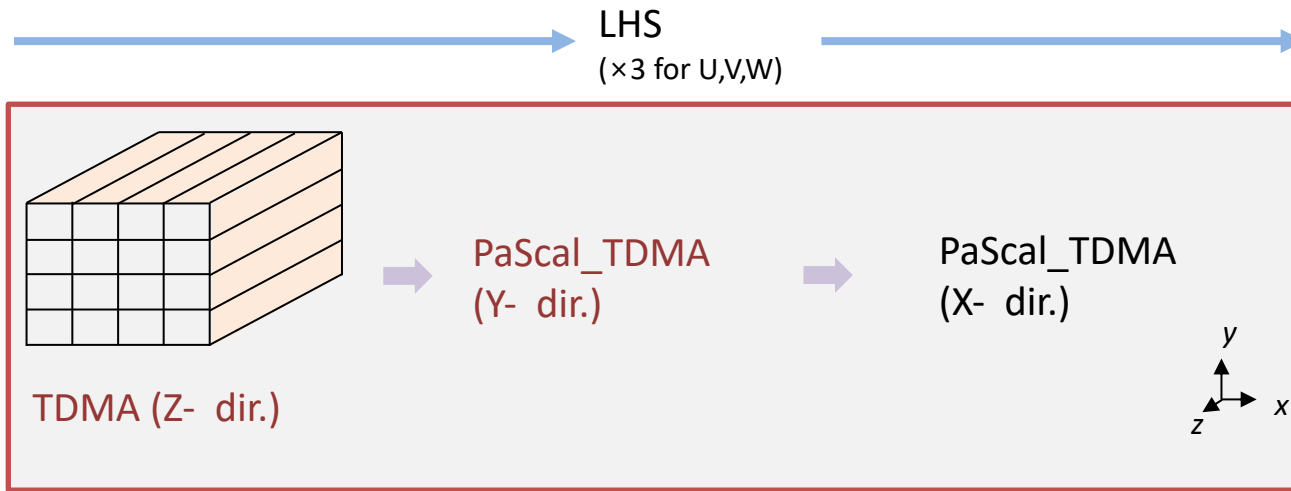
**Poisson solver** 내의 TDMA 사용 시, cache-blocking 기법을 사용해 변수 호출의 형태를 바꿈.

## 기존 LHS 계산 방식



기존 코드에서는,  
TDMA 계산을 위한 잦은 집합 통신  
(ALLTOCALL\_V)의 호출  
병렬 성능이 매우 낮았음.

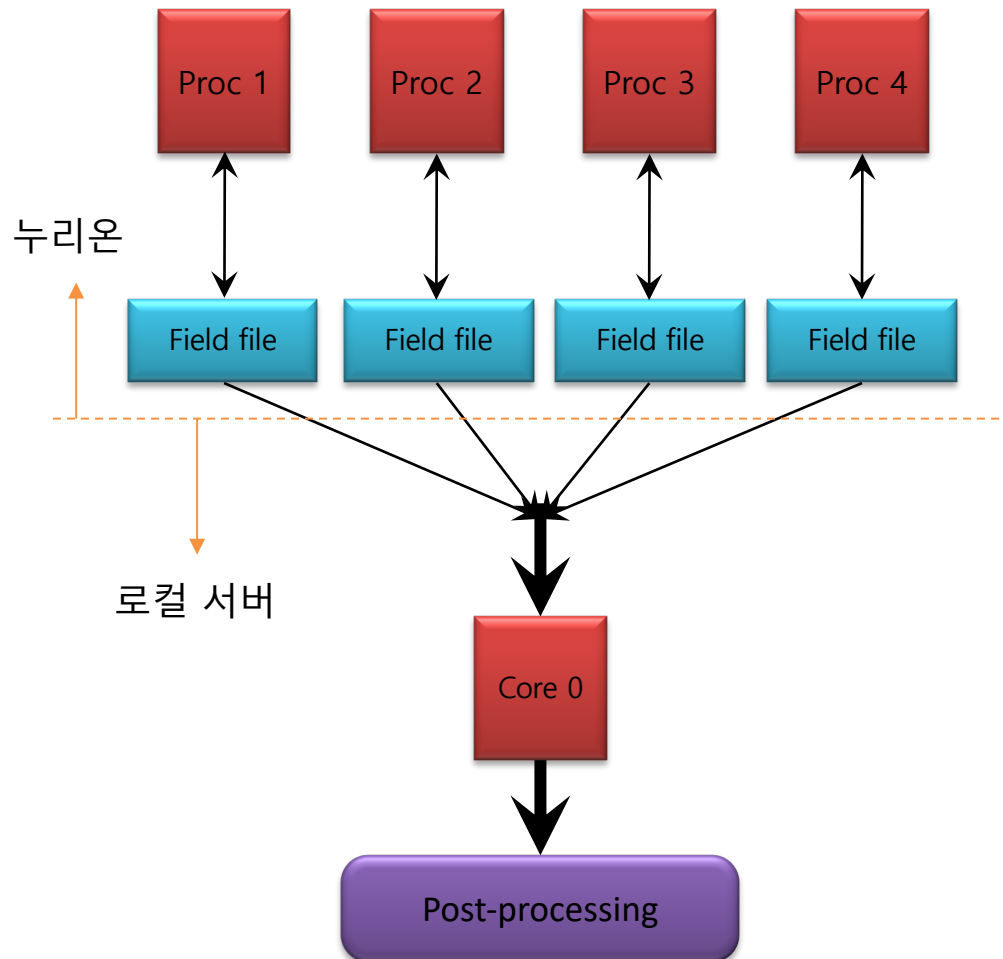
## PTDMA를 사용한 LHS 계산의 최적화



PaScal\_TDMA (Kim *et al.*, 2021)의 사용으로 각 방향 별 TDMA를 위한 집합 통신의 호출을 줄이고, 그로 인해 병렬 성능이 좋아짐.

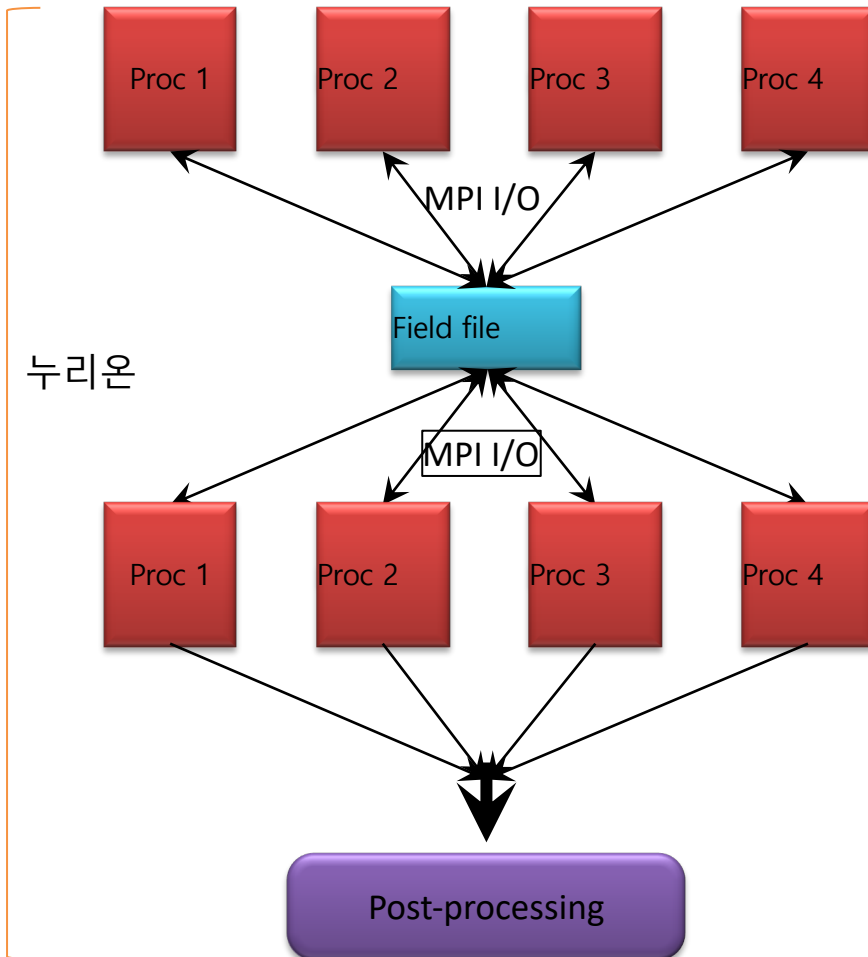


## 기존의 입출력 방식



- 각 MPI 프로세스가 각자 할당된 도메인에 해당하는 입출력 파일을 생성
- MPI 수가 늘어날수록 입출력 파일의 개수가 증가하는 문제가 발생(ex. 64노드 사용 시 최소 4,096개의 입출력 파일 생성)
- 각 입출력 파일을 로컬 서버의 한 코어가 읽어 들여, OpenMP(약 4-12 스레드)로 후처리를 진행
- 후처리 시에 한 코어가 모든 파일을 읽어 들이기 때문에 많은 시간이 소요됨
- 큰 격자를 사용할 경우, 메모리 오버헤드가 발생하는 경우가 생김

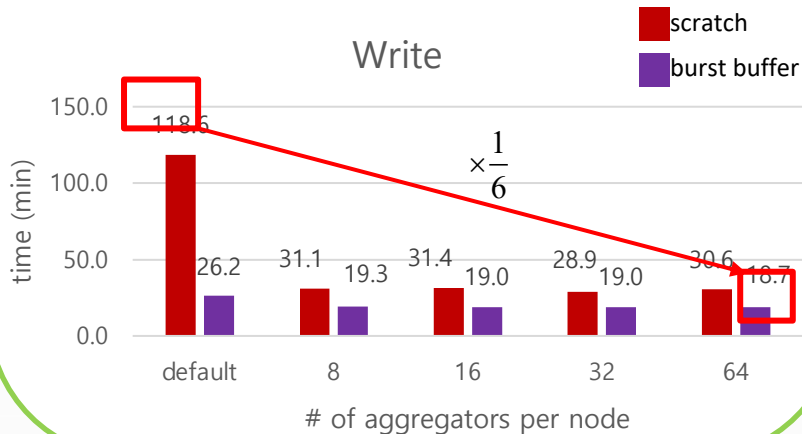
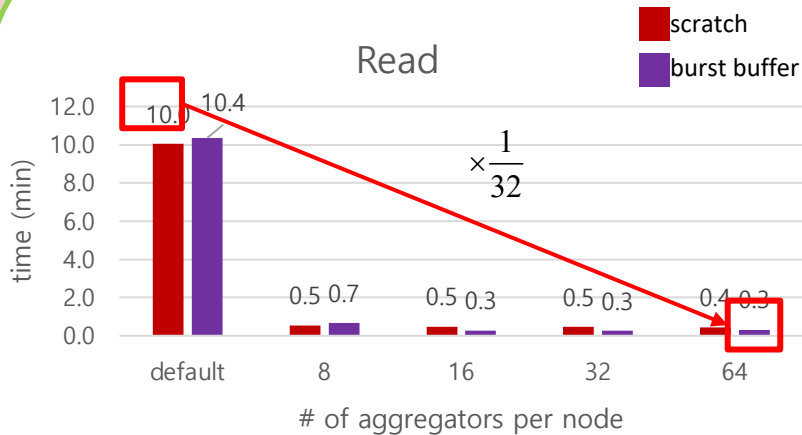
## I/O 병렬화



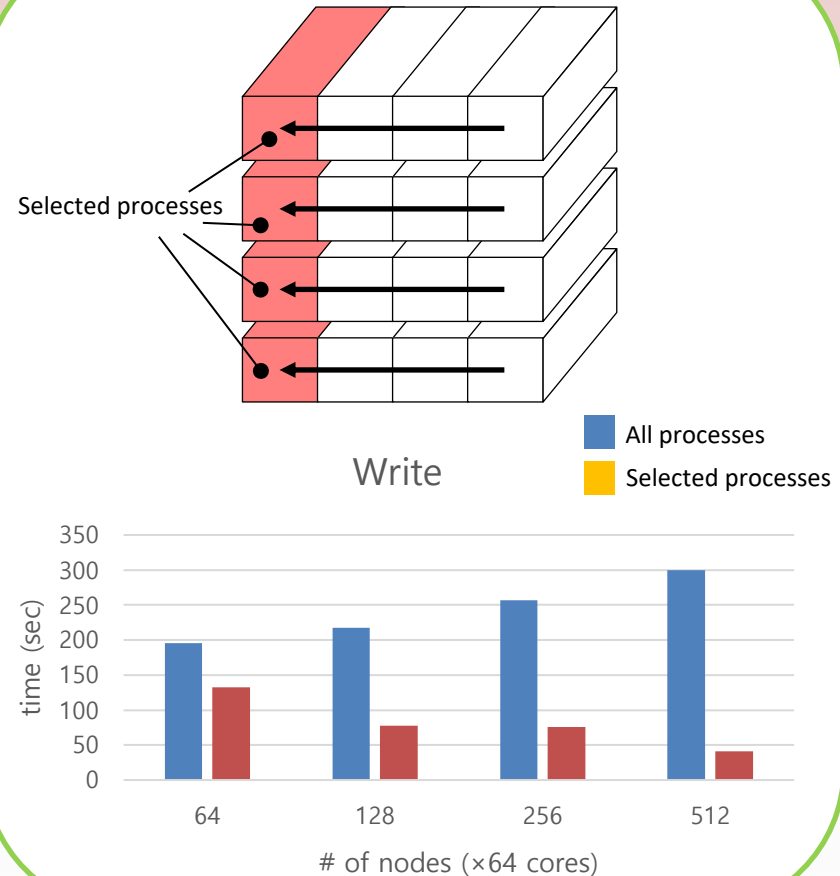
- 각 MPI 프로세스가 하나의 파일을 공유하는 병렬 입출력 방식 채택
- MPI 수가 늘어날수록 입출력 파일의 개수가 증가하는 문제를 방지함
- HDF5 기반의 CGNS (CFD General Notation System) 파일의 포맷 및 MPI I/O API를 이용
- MPI를 활용한 후처리 과정의 병렬화를 통해 효율적인 후처리 시스템 구축
- I/O 최적화를 위해 **ROMIO** 및 **버스트 버퍼 (burst buffer)**를 이용한 I/O 성능 향상

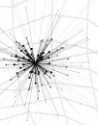
- ROMIO 및 버스트 버퍼를 활용한 I/O 병렬화를 통해 CGNS 파일을 입/출력하는 데에 소요되는 시간을 단축

- 32,768 프로세스 사용
- 128 GB 크기 CGNS 파일의 I/O



- 사용 노드가 많아짐에 따라 I/O의 부하가 심해지는 문제가 있음
- 각 노드의 대표 프로세스가 I/O를 담당하도록 수정하여 부하를 해소





Q&A

감사합니다.