Introduction to the KISTI's application support activities

2016.02.17.

Ji-Hoon Kang (jhkang@kisti.re.kr)
Dept. of Supercomputing Application
Supercomputing Service Center

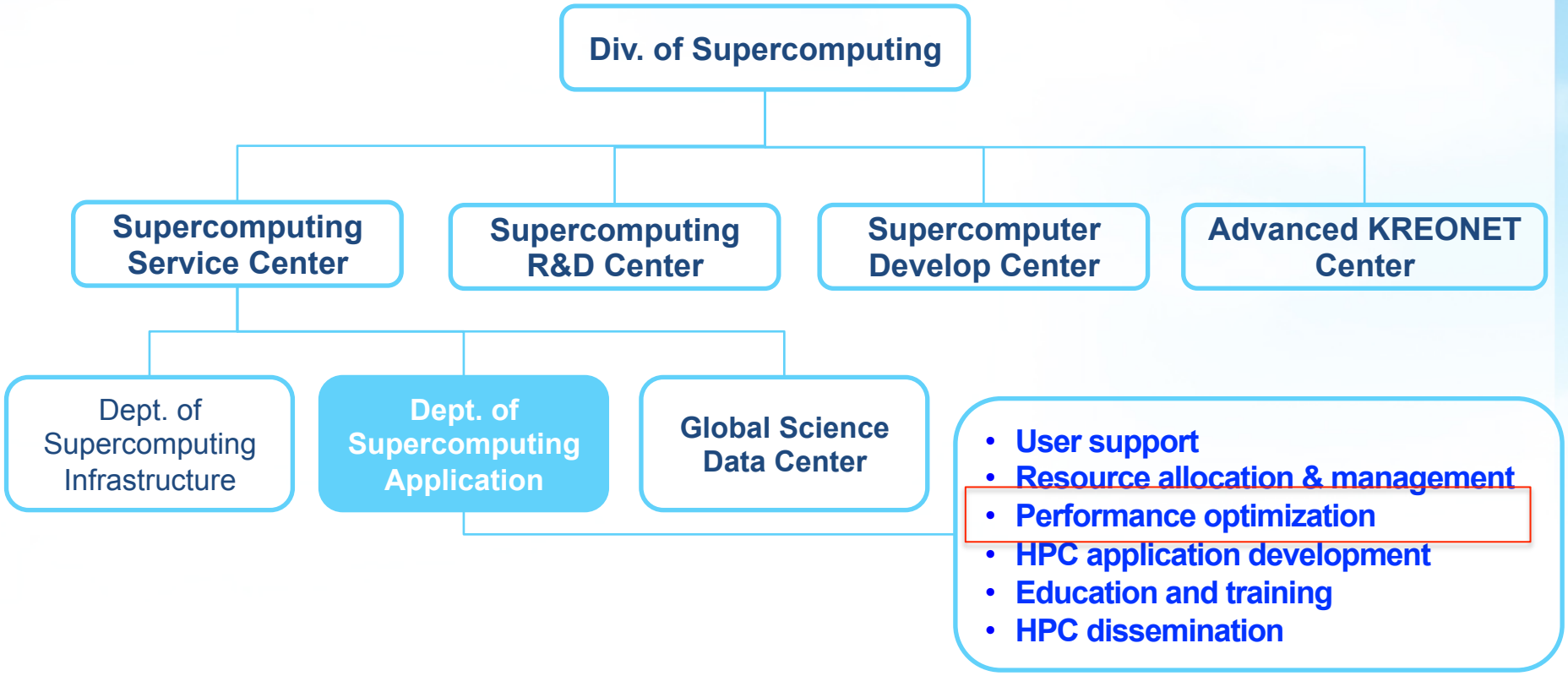
KiSTi 한국과학기술정보연구원 Korea Institute of Science and Technology Information www.kisti.re.kr

University of Tsukuba 筑波大学

Contents

- **Application support in KISTI**
  - Organization
  - Parallelization projects
  - Two case studies

Department of supercomputing application



Computer Scientists
Jeong, Yosang
Kwon, Oh-kyoung
Lee, Seungmin
Myung, Hunjoo
Park, Sangbae

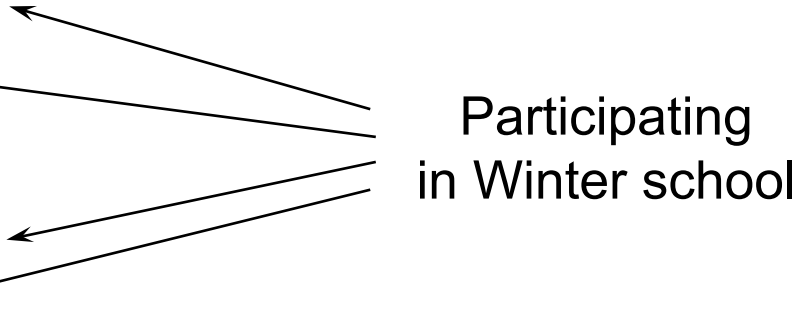
+

Domain Scientists
Jeong, Heejin (Civil. Eng.)
Jeong, Minjoong (Mech. Eng.)
Kang, Jihoon (Mech. Eng.)
Lee, Hongsuk (Physics)
Ryu, Hoon (Elec. Eng.)
Yeom, Minsun (Chem. Eng.)

Mores on parallelization and optimization

- **Purposes**
  - Provides high-level support to research scientists using KISTI supercomputer for the development of parallelized and optimized applications in their research fields
  - Aims the development of parallel algorithm and optimization technique targeting to the real scientific user applications.

- **Deeply involved members**
  - **Jeong, Yosang (Computer Science)**
  - **Kang, Ji-Hoon (Mechanical Eng.)**
  - Kwon, Oh-kyung (Computer Science)
  - **Lee, Seungmin (Computer Science)**
  - **Ryu, Hoon (Electronic Eng.)**



Parallelization project in 2014

	Co-work affiliation	Field	Platform	Improvement	Note
1	Gachon Univ.	Medical	Tachyon	2.39x @16cores	MPI Parallelization
2	KAIST	Mechanical (Turbulence)	Tachyon	Scale up to 4,096cores	MPI Parallelization
3	KAIST	Chemistry	Tachyon	66.4x @512cores	MPI Parallelization
4	Yousei Univ.	Weather	Tachyon	2.1x , (scale up to 4,096 cores)	MPI Parallelization
5	KIAS	Astrophysics	Tachyon	Scale up to 8,000cores	Perf. Optimization
6	KATECH	Electric Wave	GAIA	76.4x @64cores	OpenMP Parallelization
7	KAI CO.	Mechanical(CFD)	Tachyon	55x@256cores	MP Parallelization
8	KOIST	Ocean	GAIA	7.2x	OpenMP Parallelization
9	Soon Chun Hyang Univ. Hospital	Medical	Tachyon	326x(512cores)	MPI Parallelization
10	KARI	Satellite Image	GAIA	2.1x	Perf. Optimization

	Co-work affiliation	Field	Platform	Tesla	Xeon Phi
1	KIAS	Astrophysics	GPU	4.3x speedup	-
2	Kyungwon Tech.	Electronic Eng.	GPU	3x speedup	-
3	KAIST	Nanomechanics	GPU/Xeon Phi	4.16x	1.04x
4	Seoul Natl. Univ.	Lattice QCD	GPU/Xeon Phi	7.3x	1.4x speedup

Parallelization project in 2015

	Co-work affiliation	Field	Code description	Implementation	Results
1	Hanbat univ.	Mech. Eng.	CFD	MPI+OpenMP	Scale up to 4,096 cores using 2D domain decomposition
2	KISTI	Elec. Eng.	Electronic structure solver	MPI+OpenMP / Xeon Phi	Scale up to 5,120 cores / 1.3x @ KNC
3	KISTI	Mech. Eng.	SPH (N-body)	OpenMP	2.92x @ 8 OpenMP
4	KAIST	Chem.	Real-space DFT	MPI+OpenMP / Xeon Phi & GPU	Scale up to 4,608 cores / 1.7x and 3x @ KNC and GPU
5	KAIST	Nanomech	Phase field solver	GPU	1.87x (compared to 8-cores) 11x (single core)
6	Hongik Univ.	Civil. Eng.	Inverse analysis	MPI	12x @ 32 cores Stability improve by preconditioning
7	KAIST	Mech. Eng.	CFD	MPI+OpenMP	Scale up to 6,144 cores 19% perf. improve
8	Soongsil Univ.	Physics	Nuclear physics	MPI	576x @ 64 cores

MPI and OpenMP implemented mainly on Tachyon2
Xeon Phi & GPU implemented mainly on KAT (KISTI Accelerator Testbed)

Case1 : RS-DFT parallelization

Poisson solver parallelization on CPU/GPU/Xeon Phi

Oh-Kyung KWON, okkwon@kisti.re.kr
Sunghwan CHOI, sunghwan.choi@kaist.ac.kr

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

University of Tsukuba 筑波大学

RS-DFT (DFT-based Quantum chemistry package)

- **Implementation of interpolation scaling function to solve the Poisson equation**
  - The original second order partial differential equation can be reformulated as Integral form

$$\nabla^2 v(\boldsymbol{r}) + \frac{\rho(\boldsymbol{r})}{\epsilon_0} = 0 \quad \Rightarrow \quad v(\boldsymbol{r}) = \int d\boldsymbol{r}' \frac{1}{4\pi\epsilon_0} \frac{\rho(\boldsymbol{r}')}{|\boldsymbol{r} - \boldsymbol{r}'|},$$

  - By substitution, singularity would be eliminated. And

$$\int_{-\infty}^{\infty} \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \rho(\mathbf{r}_2) dr_2 = \frac{2}{\sqrt{\pi}} \int_0^{\infty} \int_{-\infty}^{\infty} e^{-t^2(\mathbf{r}_1 - \mathbf{r}_2)^2} \rho(\mathbf{r}_2) d\mathbf{r}_2 dt.$$

  - Electron density can be expanded to basis functions

$$\rho(\mathbf{r}_2) = \sum_{\alpha\beta\gamma} d_{\alpha\beta\gamma} \chi_\alpha(x_2) \chi_\beta(y_2) \chi_\gamma(z_2),$$

  - By decomposing 3D integral into 1D integrals by the seperability of Gaussian kernel, final equation would be

$$v_{\alpha\beta\gamma} = \sum_{p}^{p_{\max}} w_p \sum_{\gamma'}^{N_z} F_{\gamma\gamma'}^{z,p} \sum_{\beta'}^{N_y} F_{\beta\beta'}^{y,p} \sum_{\alpha'}^{N_x} F_{\alpha\alpha'}^{x,p} d_{\alpha'\beta'\gamma'} \quad \text{,where} \quad F_{\alpha\alpha'}^{x,p} = \int_{-\infty}^{\infty} e^{-t_p^2 (x_\alpha - x_2)^2} \chi_{\alpha'}(x_2) dx_2$$

Parallelization using MPI and OpenMP (I)

- **Algorithm 1**
  - In order to efficiently calculate nested for-statements, summation is reformulated to linear algebraic operation

MPI

$$v_{\alpha\beta\gamma} = \sum_{p}^{p_{\max}} w_p \sum_{\gamma'}^{N_z} F_{\gamma\gamma'}^{z,p} \sum_{\beta'}^{N_y} F_{\beta\beta'}^{y,p} \sum_{\alpha'}^{N_x} F_{\alpha\alpha'}^{x,p} d_{\alpha'\beta'\gamma'}$$

OpenMP

Two loops are parallelized using MPI and OpenMP

Algorithm 1 CPU algorithm (1)

1: Initialize $d_{\alpha'\beta'}^{\gamma'}$ and $F^{x,t_p}, F^{y,t_p}, F^{z,t_p}$
2: **for all** i **do** (MPI parallel)
3: **for all** γ' **do** (OpenMP parallel)
4: $T_{\alpha,\beta'}^{\gamma'} \leftarrow \mathrm{DGEMM}\left(F_{\alpha\alpha'}^{x,i}, d_{\alpha'\beta'}^{\gamma'}\right)$
5: $T_{\alpha,\beta}^{\gamma'} \leftarrow \mathrm{DGEMM}\left(T_{\alpha\beta'}^{\gamma'}, F_{\beta\beta'}^{y,i}\right)$
6: **end**
7: Transpose $T_{\alpha,\gamma'}^{\beta} \leftarrow T_{\alpha,\beta}^{\gamma'}$
8: **for all** β **do** (OpenMP parallel)
9: $T_{\alpha,\gamma}^{\beta} \leftarrow \mathrm{DGEMM}\left(T_{\alpha\gamma'}^{\beta}, F_{\gamma'\gamma}^{y,i}\right)$
10: **end**
11: Unfold $T_{\alpha,\beta,\gamma} \leftarrow T_{\alpha,\gamma}^{\beta}$
12: $V_{\alpha,\beta,\gamma} \leftarrow V_{\alpha,\beta,\gamma} + w_p T_{\alpha,\beta,\gamma}$
13: **end**

Parallelization using MPI and OpenMP (II)

- **Algorithm 2 for more parallelism**
  - Designed to use more MPI processes in order to increase scalability

OpenMP

MPI

$$v_{\alpha\beta\gamma} = \sum_{p}^{p_{\max}} w_p \sum_{\gamma'}^{N_z} F_{\gamma\gamma'}^{z,p} \sum_{\beta'}^{N_y} F_{\beta\beta'}^{y,p} \sum_{\alpha'}^{N_x} F_{\alpha\alpha'}^{x,p} d_{\alpha'\beta'\gamma'}$$

MPI

  - By accessing contiguous memory space, matrix multiplication can be replaced to dot product, from DGEMM to DDOT operation, because F matrix is a symmetric Toeplitz matrix

$$A = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdots & \cdots & a_{-(n-1)} \\ a_1 & a_0 & a_{-1} & \ddots & & \vdots \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \cdots & \cdots & a_2 & a_1 & a_0 \end{bmatrix}$$

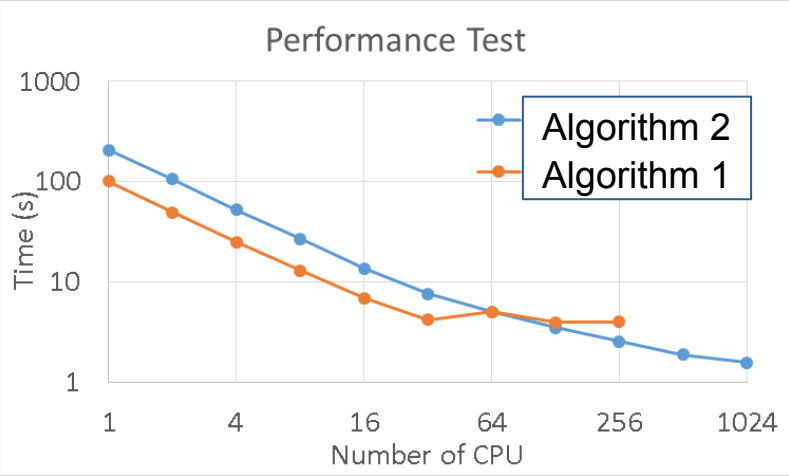
Algorithm 2 CPU algorithm (2)

```
1:  Initialize d^{γ'}_{α'β'} and F^{x,i}_{|α_max−α'|}, F^{y,i}_{|β_max−β'|}, F^{z,i}_{|γ_max−γ'|}
2:  for all t_p do (MPI parallel)
3:      for all γ' do (MPI parallel)
4:          for all β' do (OpenMP parallel)
5:              d^{β'γ'}_{α'} ← d^{γ'}_{α'β'}
6:              for all α' do
7:                  T^{γ'}_{α,β'} ← DDOT(F^{x,i}_{|α_max−α'|}, d^{β'γ'}_{α'})
                    end
8:              end
9:              for all α do (OpenMP parallel)
10:                 T^{αγ'}_{β'} ← T^{γ'}_{αβ'}
11:                 for all β' do
12:                     T^{γ'}_{α,β} ← DDOT(T^{αγ'}_{β'} F^{y,i}_{|β_max−β'|})
                    end
13:             end
14:         end
15:         for all β do (MPI parallel)
16:             for all γ' do (OpenMP parallel)
17:                 T^{αγ'}_{β'} ← T^{γ'}_{αβ}
18:                 for all γ do
19:                     T^{β}_{α,γ} ← DDOT (T^{β}_{αγ'}, F^{z,i}_{|γ_max−γ'|})
                    end
20:             end
21:         end
22:         Unfold T_{α,β,γ} ← T^{β}_{α,γ}
23:         V_{α,β,γ} ← V_{α,β,γ} + w_p T_{α,β,γ}
24: end
```

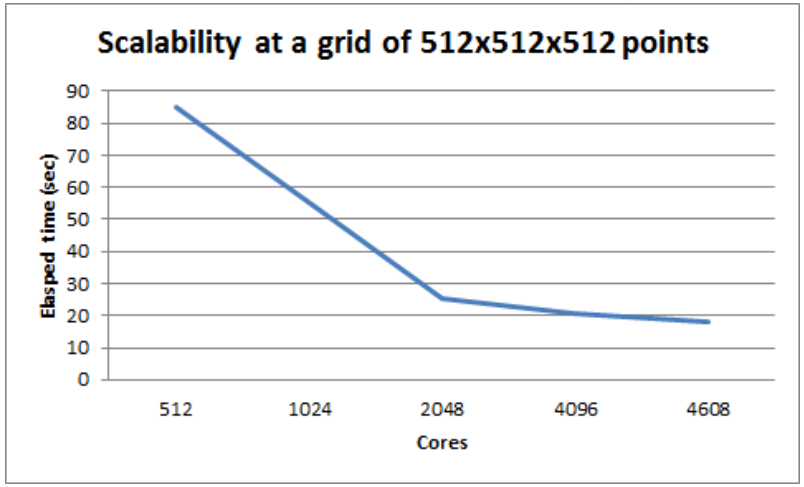
Three loops are parallelized using two-step MPIs and OpenMP

Performance results on multiple nodes

- **Performance evaluation at Tachyon2**
  - Algorithm 1 shows better performance with lower number of CPU, but algorithm 2 becomes to out perform algorithm 1 as the number of CPU increases.
  - Algorithm 2 can enjoy much more parallelism and it scales up to 4608 cores: 3.08x improvement compared to 1024 cores.
    (512x512x512 points)



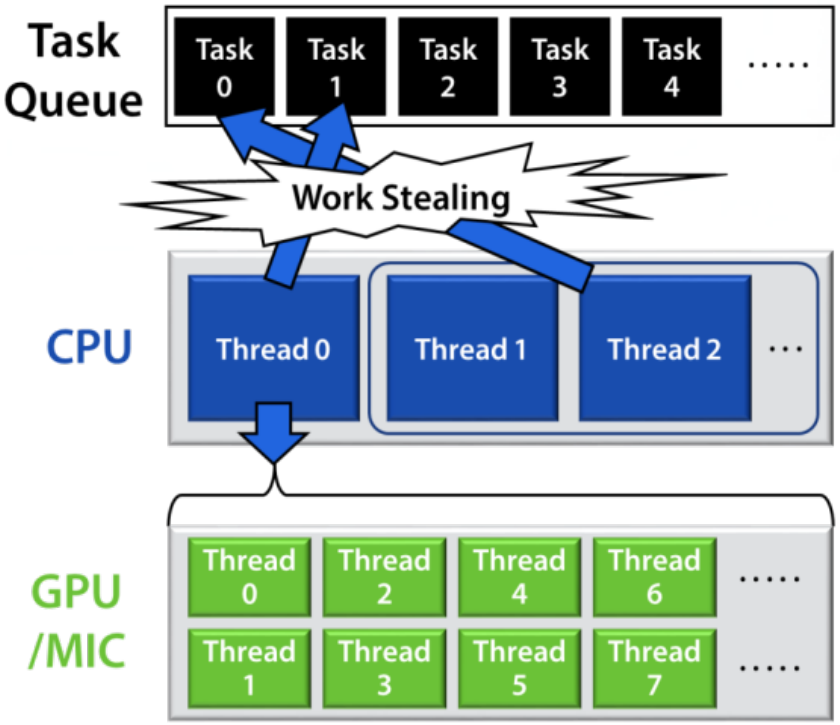
[Comparison of algorithm 1 and 2 using 259^3 (=17,373,979) number of grid points at Tachyon2]



[Scalability of algorithm 2 using 512^3 number of grid points at Tachyon2]

Parallelization on GPU or Xeon Phi single node

- **Algorithm 3 for Heterogeneous computing**
  - Work stealing algorithm
  - Thread 0 of CPU act as master. Other CPU threads and GPU (or Xeon PHI card) work as worker 1 and worker 2, respectively.
  - If we have two GPU cards, then we have three workers.



[Schematic diagram of work stealing parallelization]

Parallelization on GPU or Xeon Phi

- **Algorithm 3 for Heterogeneous computing**

Algorithm 4 Heterogeneous algorithm

1: Initialize $d^{\gamma'}_{\alpha'\beta'}$ and $F^{x,i}$,$F^{y,i}$,$F^{z,i}$
2: Asynchronosly copy $\{d^{\gamma'}_{\alpha'\beta'}\}$ from CPU to GPU
3: $i_{cpu} \leftarrow 0$, $cpuState \leftarrow 0$
4: $i_{gpu} \leftarrow 0$, $gpuState \leftarrow 0$
5: $i \leftarrow 0$
6: **while** *true* **do**
7: **if** $i \geq size\_t$ & $gpuState$=0
8: & $endCPU = true$ & $endGPU = true$ **then**
9: *break*
10: **end**
11: $endCPU \leftarrow checkCPU()$
12: **if** endCPU = *true* **then**
13: Join pthread and update $V^{cpu}_{\alpha\beta\gamma}$
14: **if** $i < size\_t$ **then**
15: $i_{cpu} \leftarrow i$
16: $i \leftarrow i+1$
17: Create $N^{thread} - 1$ pthreads with $cpu\_function$
18: **end**
19: **end**
20: $endGPU \leftarrow checkGPU()$

CPU

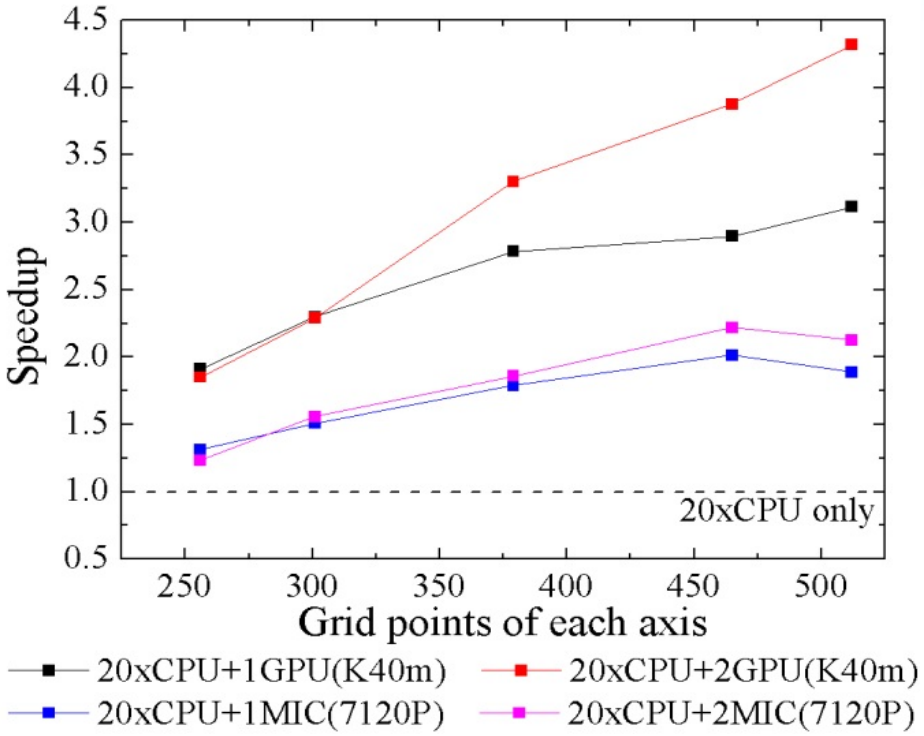
21: **if** $i < size\_t$ & $endGPU = true$ **then**
22: **if** $gpuState = 0$ **then**
23: $i_{gpu} \leftarrow i$
24: Copy $F^{x,i_{gpu}}$ from CPU to GPU
25: $\{T^{\gamma'}_{\alpha,\beta'}\} \leftarrow \text{DGEMMBatched}\left(F^{x,p}_{\alpha\alpha'}, \{d^0_{\alpha'\beta'}, d^1_{\alpha'\beta'} \cdots d^{\gamma_{max}}_{\alpha'\beta'}\}\right)$
26: $i \leftarrow i+1$, $gpuState \leftarrow gpuState + 1$
27: **end**
28: **if** $gpuState = 1$ **then**
29: Copy $F^{y,i_{gpu}}$ from CPU to GPU
30: $\{T^{\gamma'}_{\alpha,\beta}\} \leftarrow \text{DGEMMBatched}\left(\{T^0_{\alpha,\beta'}, T^1_{\alpha,\beta'} \cdots T^{\gamma_{max}}_{\alpha,\beta'}\}, F^{y,i_{gpu}}_{\alpha\alpha'}\right)$
31: $gpuState \leftarrow gpuState + 1$
32: **end**
33: **if** $gpuState = 2$ **then**
34: $\{T^{\beta}_{\alpha,\gamma'}\} \leftarrow \text{Transpose}\left(\{T^{\gamma'}_{\alpha,\beta}\}\right)$
35: $gpuState \leftarrow gpuState + 1$
36: **end**
37: **if** $gpuState = 3$ **then**
38: Copy $F^{z,i_{gpu}}$ from CPU to GPU
39: $\{T^{\beta}_{\alpha,\gamma}\} \leftarrow \text{DGEMMBatched}\left(\{T^0_{\alpha,\gamma}, T^1_{\alpha,\gamma} \cdots T^{\beta_{max}}_{\alpha,\gamma}\}, F^{z,i_{gpu}}_{\alpha\alpha'}\right)$
40: Unfold $T_{\alpha,\beta,\gamma} \leftarrow T^{\beta}_{\alpha,\beta}$
41: $V^{gpu}_{\alpha,\beta,\gamma} \leftarrow V^{gpu}_{\alpha,\beta,\gamma} + w_{i_{gpu}} * T_{\alpha,\gamma,\beta}$
42: $gpuState \leftarrow 0$
43: **end**
44: **end**
45: **end**
46: Copy $T_{\alpha,\beta,\gamma}$ from GPU to CPU
47: $V_{\alpha,\beta,\gamma} \leftarrow V^{cpu}_{\alpha,\beta,\gamma} + V^{gpu}_{\alpha,\beta,\gamma}$

GPU or PHI

4

Results on a single node

- **Running results on the Xeon phi or NVIDIA GPU node**
  - Test machine : Two 10-core Intel Xeon E5-2670 v2 with 7120p(KNC) or K40(Tesla)
  - Grid numbers : 512x512x512 points
  - NVIDIA GPU(K40): 4x(2 cards) and 3x(1 card) improvement compared to 20 CPU cores
  - Xeon Phi (7120P): 1.7x improvement compared to 20 CPU cores (512x512x512 points)



[Speedup results]

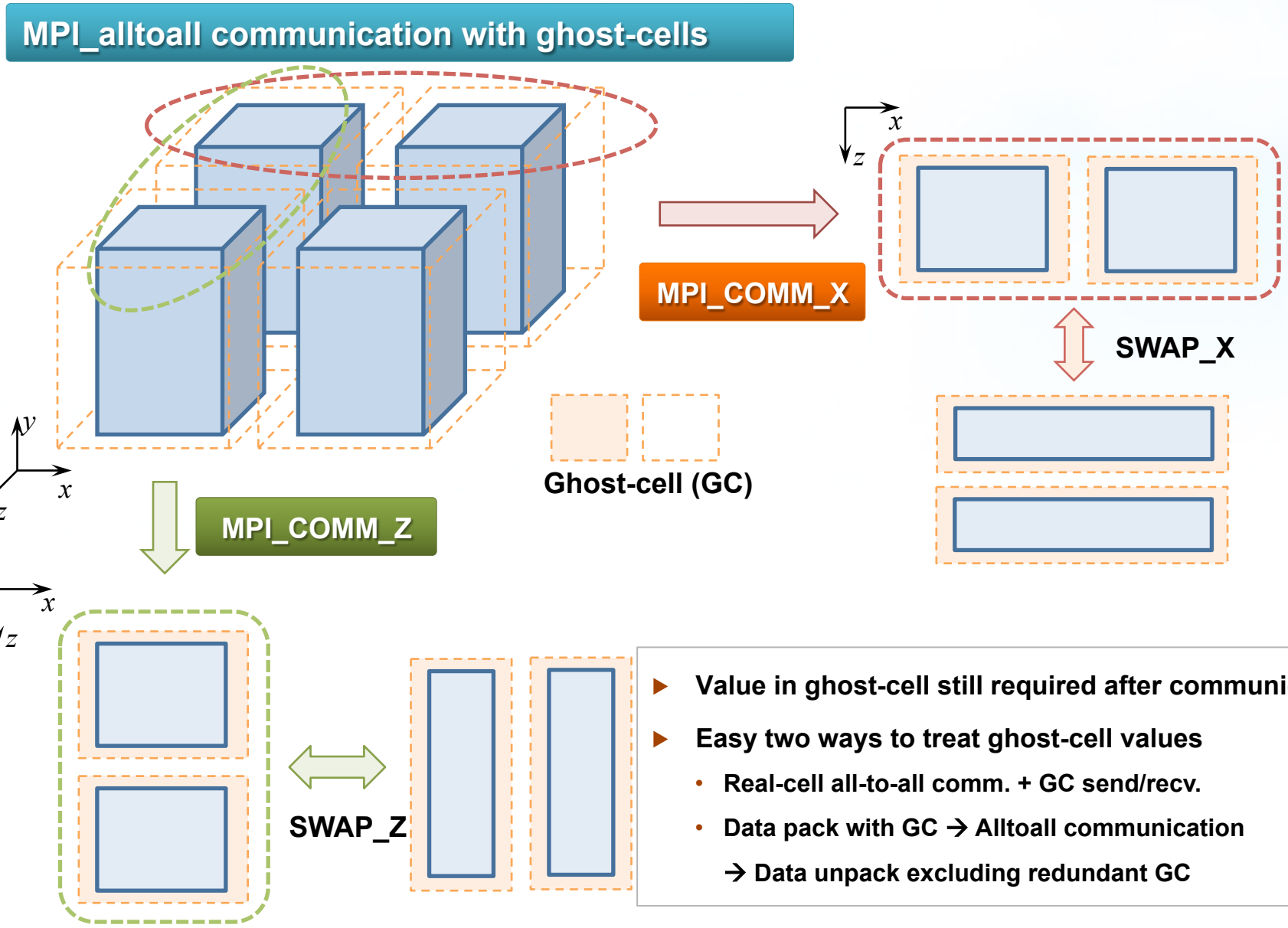
Case 2 : DNS CFD code optimization

MPI_alltoall communication improvement using DDT

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

University of Tsukuba
筑波大学

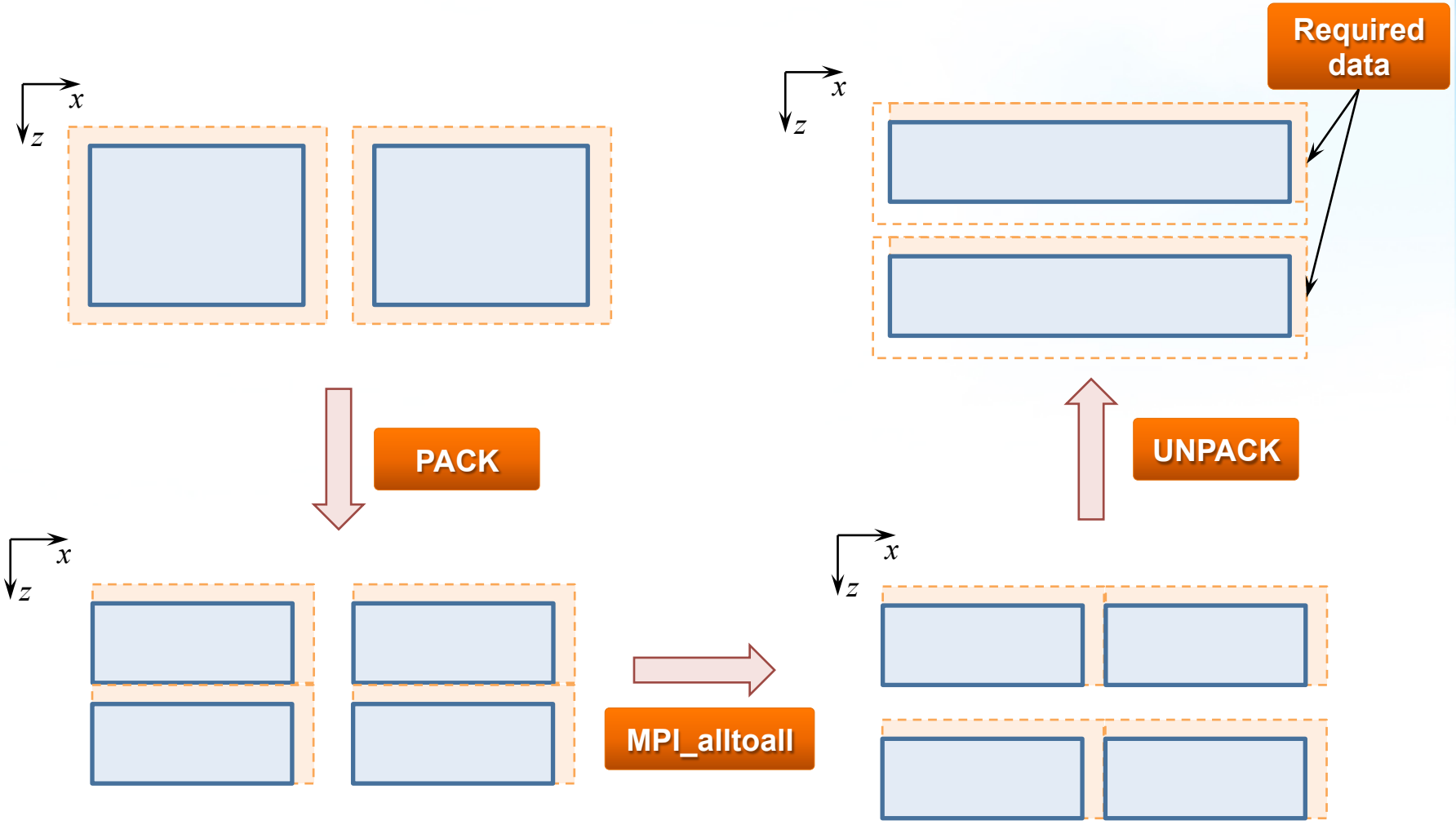
All-to-all communication in 2D domain decomposition



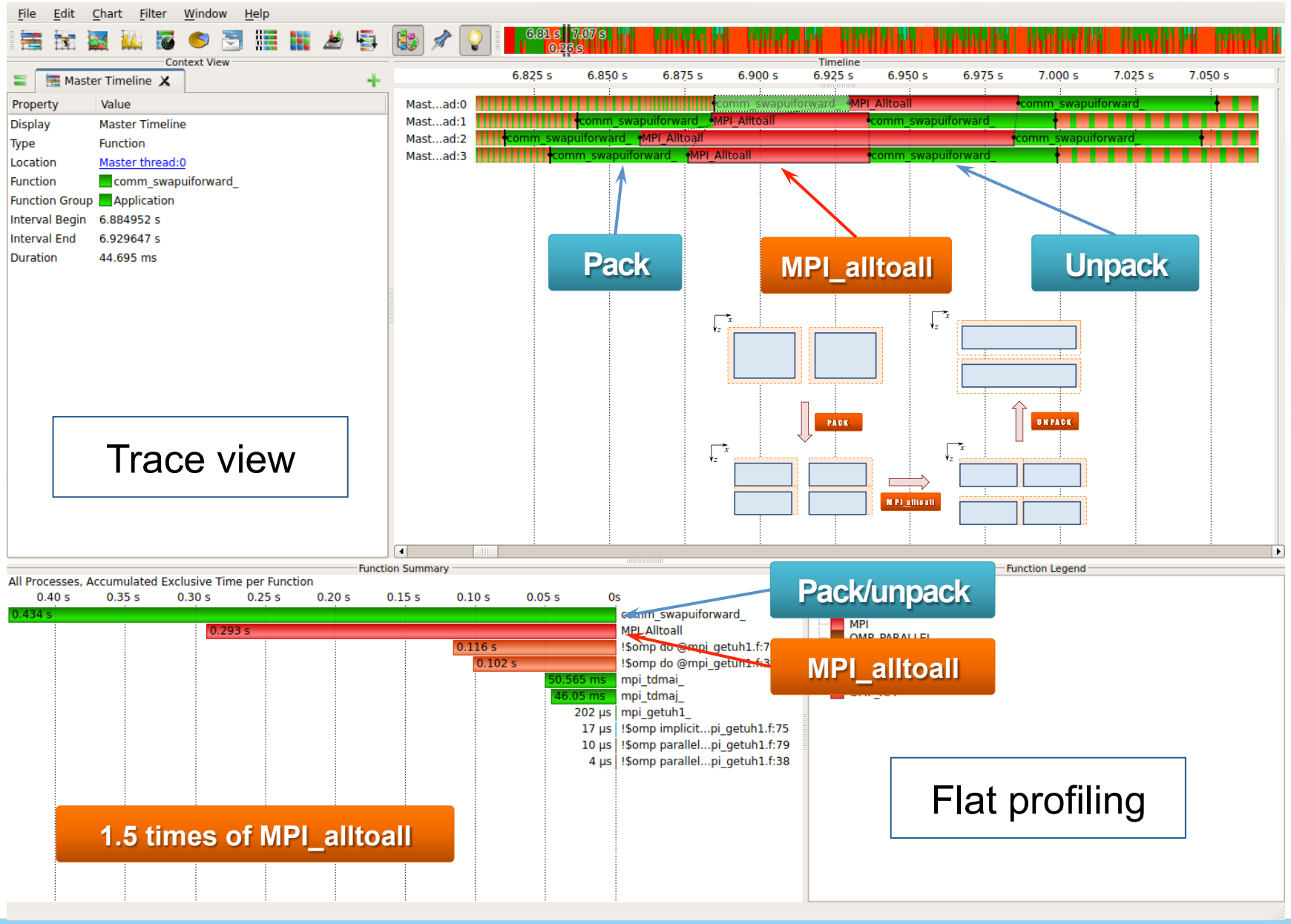
- Value in ghost-cell still required after communication
- Easy two ways to treat ghost-cell values
  - Real-cell all-to-all comm. + GC send/recv.
  - Data pack with GC → Alltoall communication
    → Data unpack excluding redundant GC

Example – all-to-all communication with GC

Data pack including GC – MPI_alltoall – Data unpack excluding redundant GC



Profiling - Pack-Unpack cost

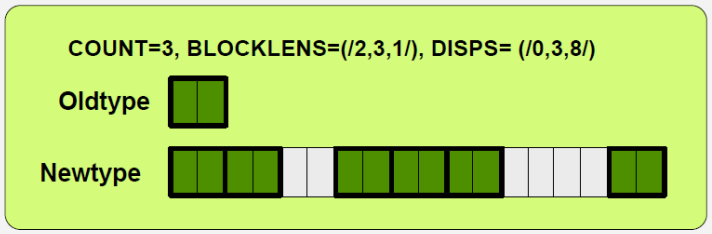


Improve

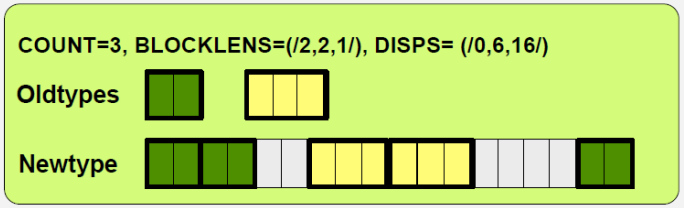
Using derived data type (DDT)

- **DDT of non-contiguous data**
  - MPI_type_indexed or MPI_type_struct
  - Indexing the first and last address of data
  - Create the data type with/without ghost-cell as required
  - Build 3-dimensional data structure with overlapping the DDT as the dimension increases

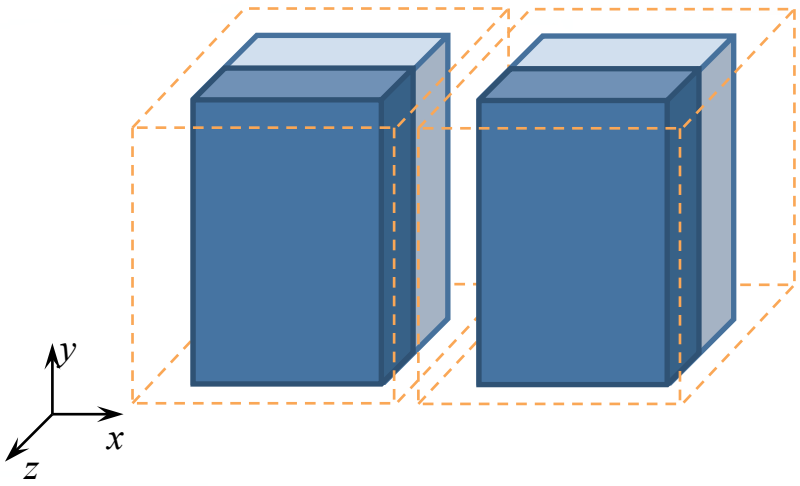
- **MPI_TYPE_INDEXED**
  - Creates a new type from blocks comprising identical elements. The size and displacements of the blocks can vary (e.g. upper triangle of a matrix)
  - F9x: MPI_TYPE_INDEXED(COUNT, BLOCKLENS, DISPS,OLDTYPE,NEWTYPE, ERR)
    - BLOCKLENS=lengths of the blocks (array)
    - DISPLS=displacements (array) in OLDTYPES
- **MPI_TYPE_CREATE_HINDEXED**
  - As MPI_TYPE_INDEXED but displacements in bytes

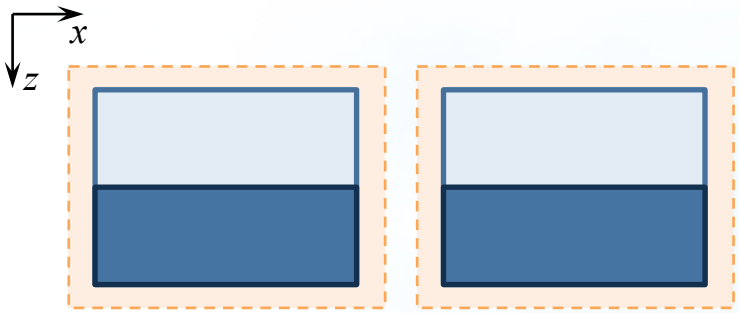


- **MPI_TYPE_CREATE_STRUCT**
  - Most general type constructor.
  - Creates a new type from heterogeneous blocks
  - E.g. Fortran 77 common blocks, Fortran 9x and C structures.
  - F9x: MPI_TYPE_STRUCT(count, array_of_blocklengths, array_of_disp,array_of_types, newtype, error)
    - count,array_of_blocklengths: as earlier (integer)
    - array_of_disp: Displacements in bytes (integer(KIND=MPI_ADDRESS_KIND)
    - array_of_types: Array of block types
- **MPI_GET_ADDRESS** can be used to calculate displacement



MPI_type_struct – SendType create

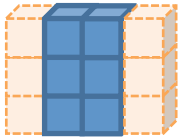




1st step

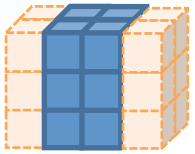
call MPI_type_get_extent &
(MPI_REAL8,dum,extent,ierr)
leng3(1)=1
leng3(2)=n1mmpi
leng3(3)=1
disp3(1)=0
disp3(2)=extent
disp3(3)=(n1mmpi+2)*extent
type3(1)=MPI_LB
type3(2)=MPI_real8
type3(3)=MPI_UB
call MPI_type_struct &
(3,leng3,disp3,type3,ddt_fw1,ierr)

2nd step



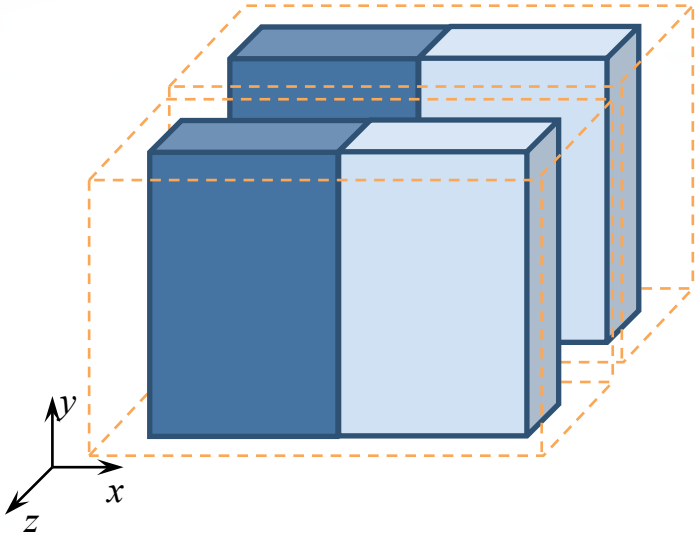
call MPI_type_get_extent &
(ddt_fw1,dum,extent,ierr)
leng2(1)=n2+1
leng2(2)=1
disp2(1)=0
disp2(2)=(n2+1)*extent
type2(1)=ddt_fw1
type2(2)=MPI_UB
call MPI_type_struct &
(2,leng2,disp2,type2,ddt_fw2,ierr)

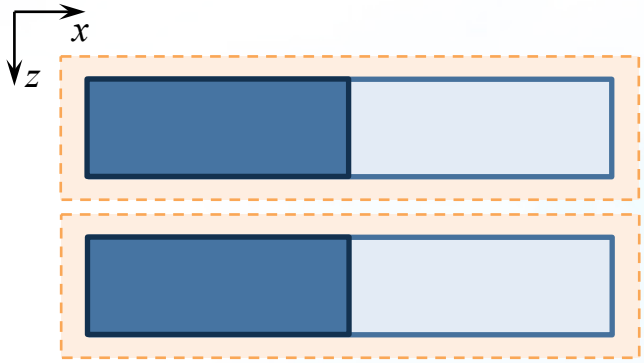
3rd step



call MPI_type_get_extent &
(ddt_fw2,dum,extent,ierr)
leng2(1)=n3mmpiblk
leng2(2)=1
disp2(1)=0
disp2(2)=n3mmpiblk*extent
type2(1)=ddt_fw2
type2(2)=MPI_UB
call MPI_type_struct &
(2,leng2,disp2,type2,ddt_swap_ux_fw,ierr)

MPI_type_struct – RecvType

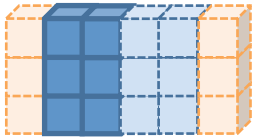




1st step

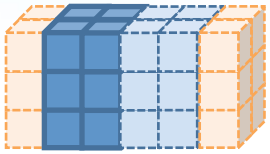
call MPI_type_get_extent &
(MPI_REAL8,dum,extent,ierr)
leng3(1)=1
leng3(2)=n1mmpi
leng3(3)=1
disp3(1)=0
disp3(2)=extent
disp3(3)=(n1m+2)*extent
type3(1)=MPI_LB
type3(2)=MPI_real8
type3(3)=MPI_UB
call MPI_type_struct &
(3,leng3,disp3,type3,ddt_bw1,ierr)

2nd step



call MPI_type_get_extent &
(ddt_bw1,dum,extent,ierr)
leng2(1)=n2+1
leng2(2)=1
disp2(1)=0
disp2(2)=(n2+1)*extent
type2(1)=ddt_bw1
type2(2)=MPI_UB
call MPI_type_struct &
(2,leng2,disp2,type2,ddt_bw2,ierr)

3rd step

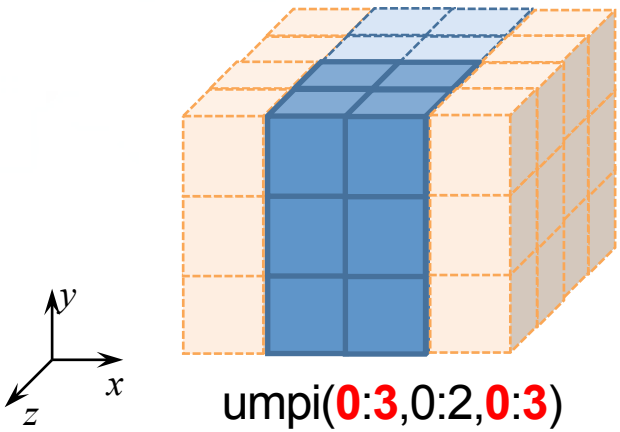


call MPI_type_get_extent &
(ddt_bw2,dum,extent,ierr)
leng2(1)=n3mmpiblk
leng2(2)=1
disp2(1)=0
disp2(2)=n3mmpiblk*extent
type2(1)=ddt_bw2
type2(2)=MPI_UB
call MPI_type_struct &
(2,leng2,disp2,type2,ddt_swap_ux_bw,ierr)

Extent of datatypes

- **Defines how a sequence of elements are laid in memory**
  - The distance between subsequent elements
  - Important to understand to send/recv more than one element of a user defined type
  - MPI_TYPE_CREATE_RESIZED is used to create a new type with user defined extent

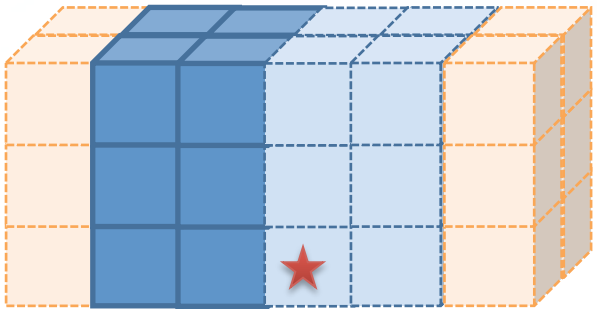
SendType



umpi(**0**:**3**,0:2,**0**:**3**)

- Address of next element = 4 x 3 x 2
  - It is same with the extent of ddt_swap_ux_fw
  - No need to define extent again

RecvType

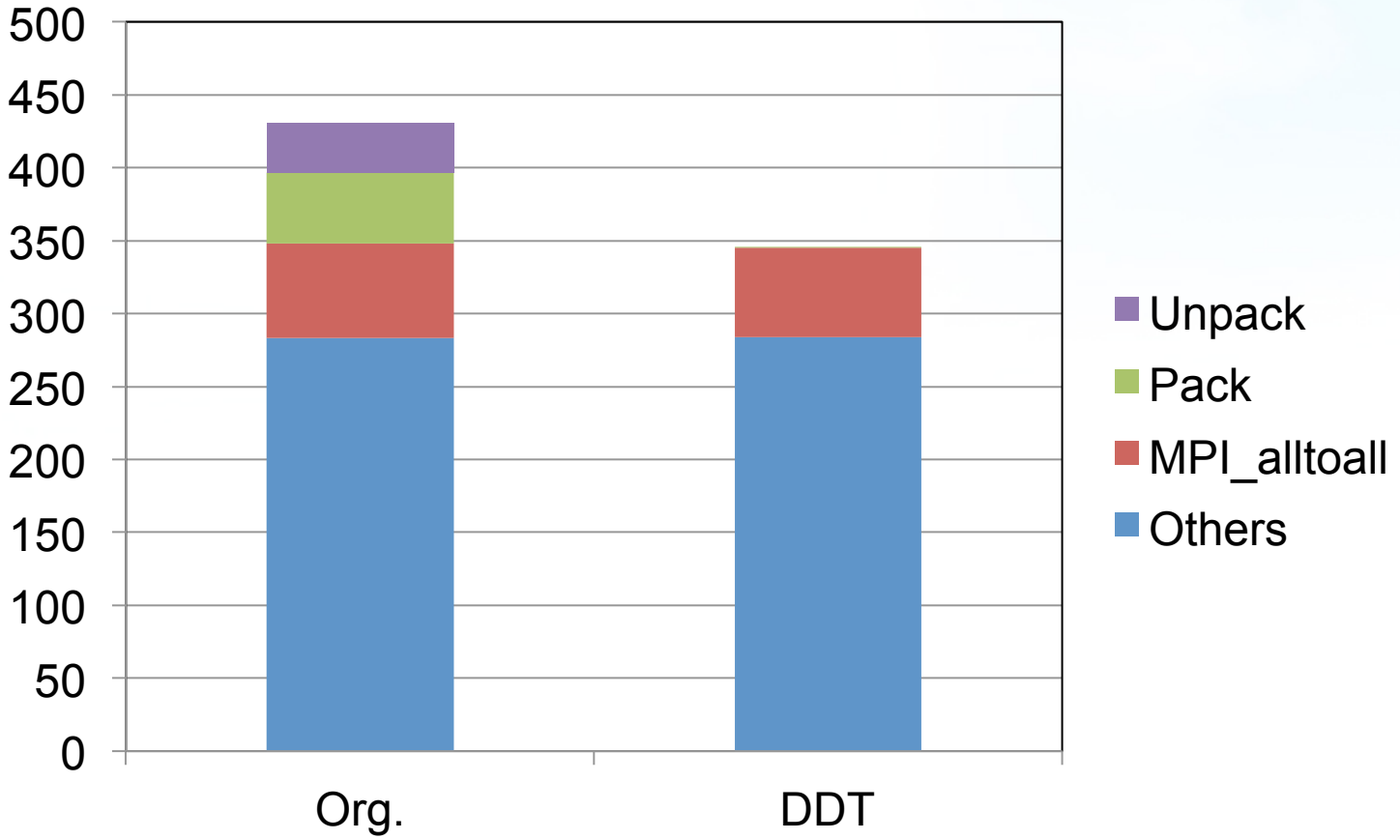


uimpi(**0**:**5**,0:2,**0**:**1**)

- Address of next element = 3
  - It is different from the extent of ddt_swap_ux_bw
  - Need to define extent again

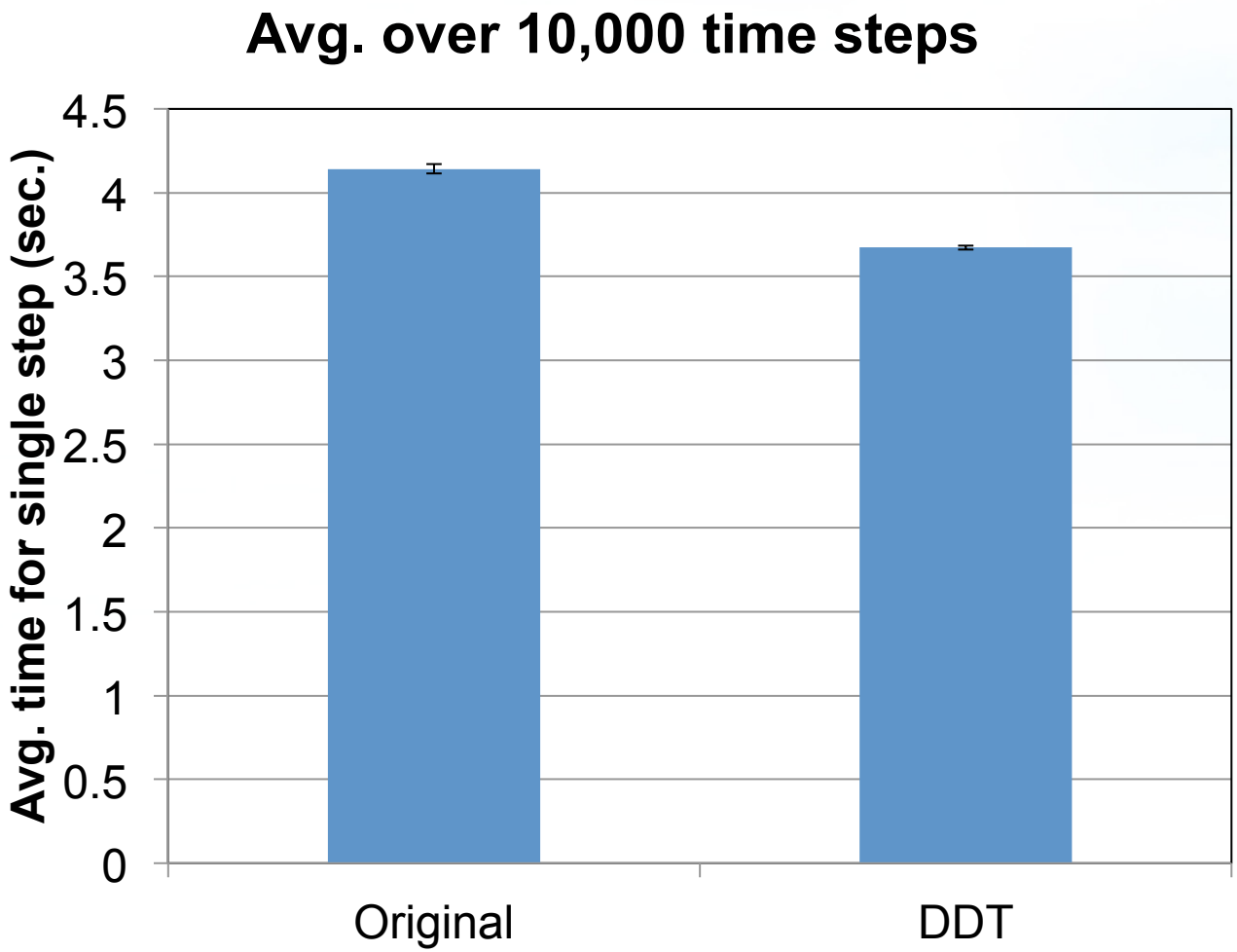
Small problem comparison

- Grid sizes 1025 x 191 x 257
- 4 MPI processes
- Running on single node of Tachyon 2



19% Overall performance improvement

Large problem running time comparison



11% Overall performance improvement