XcalableMP and XcalableACC for Productivity and Performance in HPC Challenge Award Competition

Masahiro Nakao,[†‡] Hitoshi Murai,[†] Hidetoshi Iwashita[†]
Takenori Shimosaka,[†] Akihiro Tabuchi,[*] Taisuke Boku[‡*]
Mitsuhisa Sato[†‡*]

† RIKEN Advanced Institute for Computational Science, Japan
‡ Center for Computational Sciences, University of Tsukuba
* Graduate School of Systems and Information Engineering, University of Tsukuba

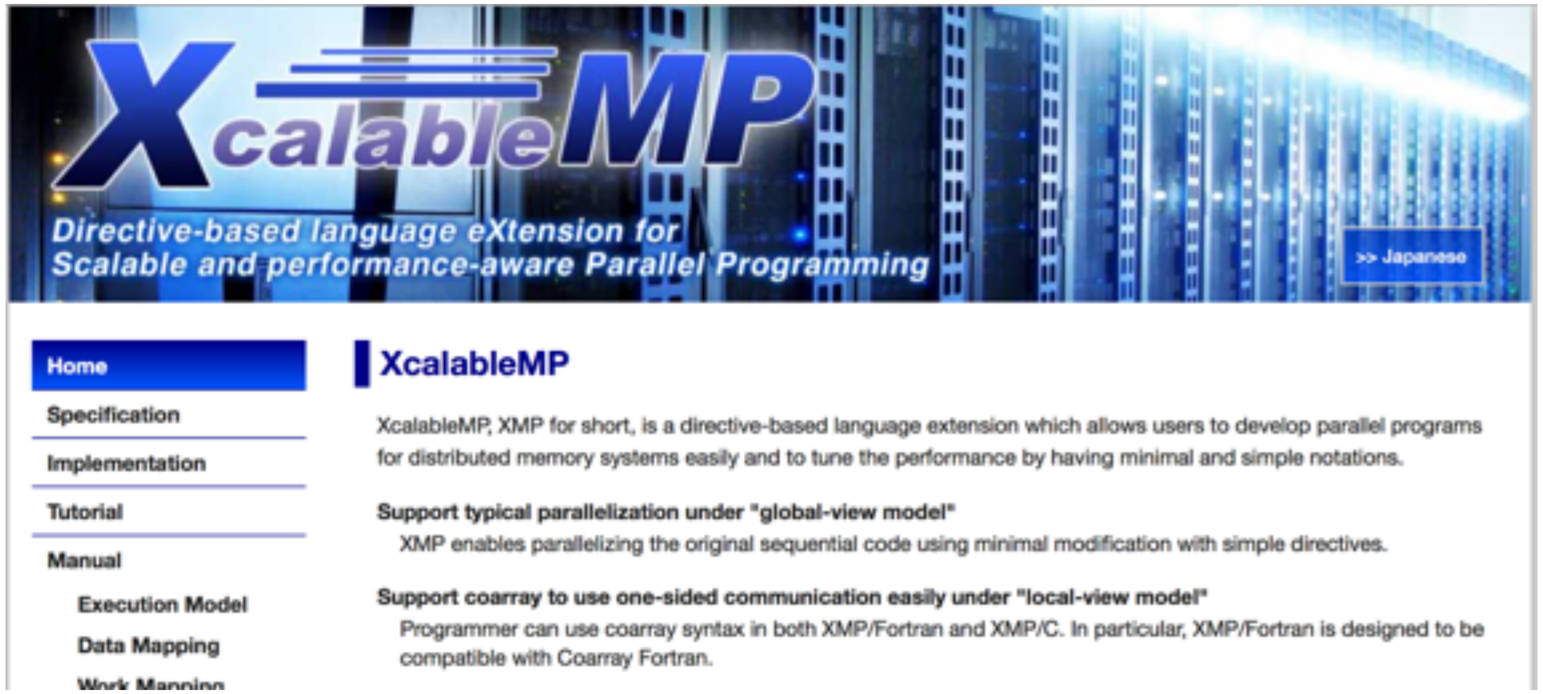
Outline

1. XcalableMP (XMP) for cluster systems (14min.)
2. XcalableACC (XACC) for accelerator cluster systems (6min.)

   Extension of XMP using OpenACC

Sorry !!, work-in-progress

The submission report is available at ***http://xcalablemp.org***



XcalableMP

XcalableMP, XMP for short, is a directive-based language extension which allows users to develop parallel programs for distributed memory systems easily and to tune the performance by having minimal and simple notations.

Support typical parallelization under "global-view model"

XMP enables parallelizing the original sequential code using minimal modification with simple directives.

Support coarray to use one-sided communication easily under "local-view model"

Programmer can use coarray syntax in both XMP/Fortran and XMP/C. In particular, XMP/Fortran is designed to be compatible with Coarray Fortran.

What is XcalableMP (XMP) ?

Directive-based language extensions of Fortran and C

- By XMP specification working group of PC cluster consortium (SC Booth#2924)
- Version 1.2.1 specification available (***http://xcalablemp.org***)

Support two memory models

- Global-view (HPF-like data/work mapping directives)
- Local-view (coarray)

Implementation of Compiler

- Omni XMP Compiler version 0.9 (***http://omni-compiler.org***)
- Platforms: Fujitsu the K computer and FX10, Cray XT/XE,
  IBM BlueGene, NEC SX, Hitachi SR, Linux clusters, etc.

Code example (Global-view)

```
int a[MAX];
#pragma xmp nodes p(4)
#pragma xmp template t(0:MAX-1)
#pragma xmp distribute t(block) on p
#pragma xmp align a[i] with t(i)

main(){
  int i, j, res = 0;

#pragma xmp loop on t(i) reduction(+:res)
  for(i = 0; i <MAX; i++){
    a[i] = func(i);
    res += array[i];
  }
```

Data distribution

add to the serial code : incremental parallelization

Work mapping and data synchronization

Code example (Local-view)

```
double a[100]:[*], b[100]:[*];
int me = xmp_node_num();
```

Define Coarrays

```
if(me == 2)
   a[:]:[1] = b[:];
```

Put Operation

```
if(me == 1)
   a[0:50] = b[0:50]:[2];
```

Get Operation

Coarray synax in XMP/C

```
array_name[start:length]:[node_number];
```

XMP/Fortran is upward compatible with Fortran 2008

Results and Machine

Summary

Four HPCC Benchmarks

Benchmark		# Nodes	Performance (/peak)	SLOC
HPL	Ver. 1	16,384	971 TFlops (46.3%)	313
	Ver. 2	4,096	423 TFlops (**80.7%**)	426
FFT		**82,944**	212 TFlops (2.0%)	205
STREAM		**82,994**	3,583 TB/s (67.5%)	69
RandomAccess		16,384	254 GUPs	253

The K computer: 82,944 nodes

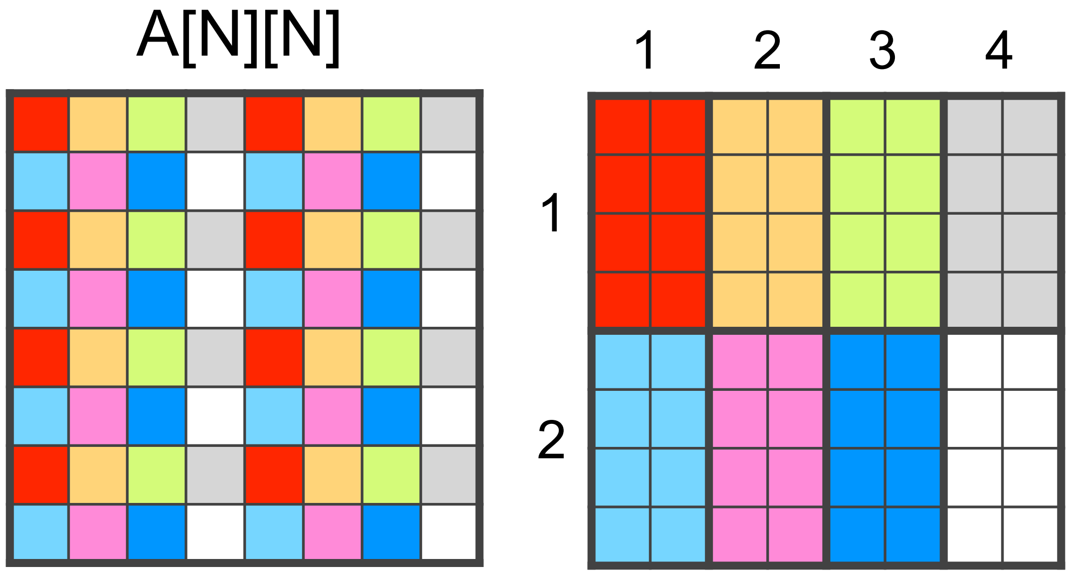
http://www.aics.riken.jp/jp/outreach/photogallery.html

- SPARC64 VIIIfx Chip, 128 GFlops
- DDR3 SDRAM 16GB, 64GB/s
- Tofu Interconnect
  - 6D mesh/torus network
  - 5GB/s x 4links x 2

HPL version 1

- Source lines of Code (SLOC) is **313**, written in XMP/C
- Block-Cyclic Distribution

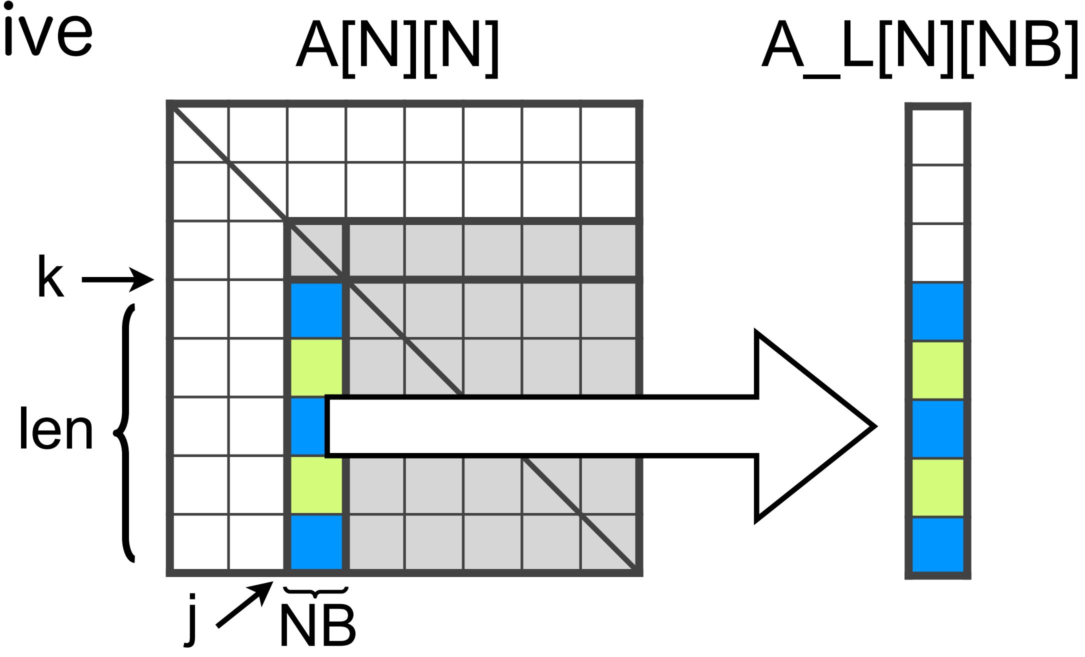
```
double A[N][N];
#pragma xmp nodes p(P,Q)
#pragma xmp template t(0:N-1, 0:N-1)
#pragma xmp distribute t(cyclic(NB), \
                   cyclic(NB)) onto p
#pragma xmp align A[i][j] with t(j,i)
```



Programmer can use BLAS for distributed array.

- Panel Broadcast by using **gmove** directive

```
double A_L[N][NB];
#pragma xmp align A_L[i][*] with t(*,i)
    :
#pragma xmp gmove
A_L[k:len][0:NB] = A[k:len][j:NB];
```

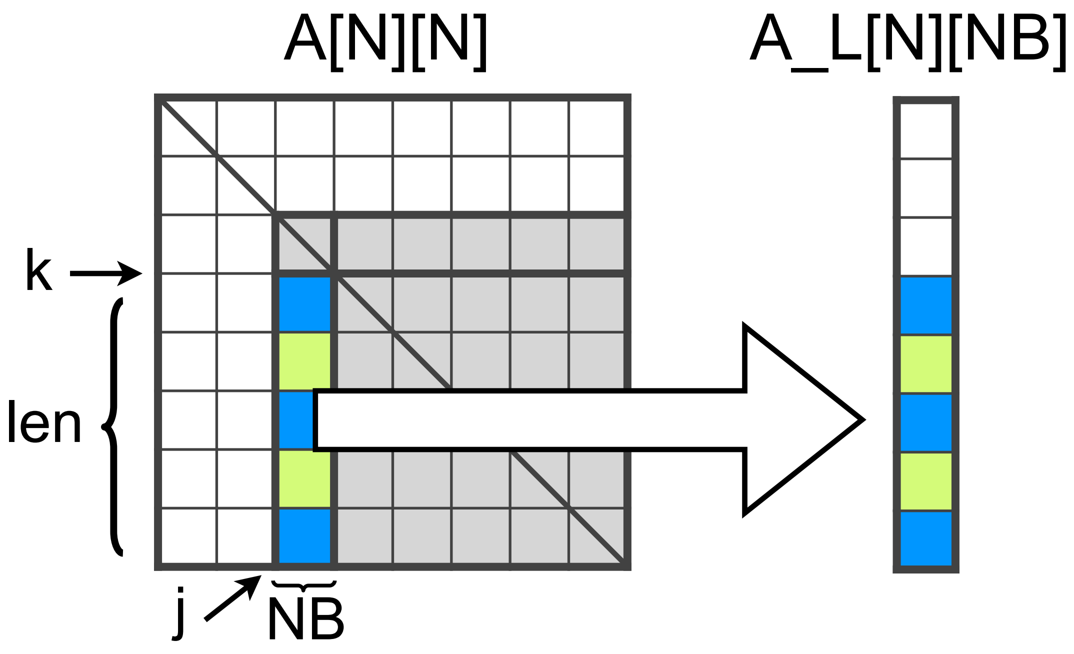


HPL version 2

- SLOC is **426**, written in XMP/C
- ”Lookahead algorithm” by using **gmove** directive with **async** clause

Overlap communication and calculation

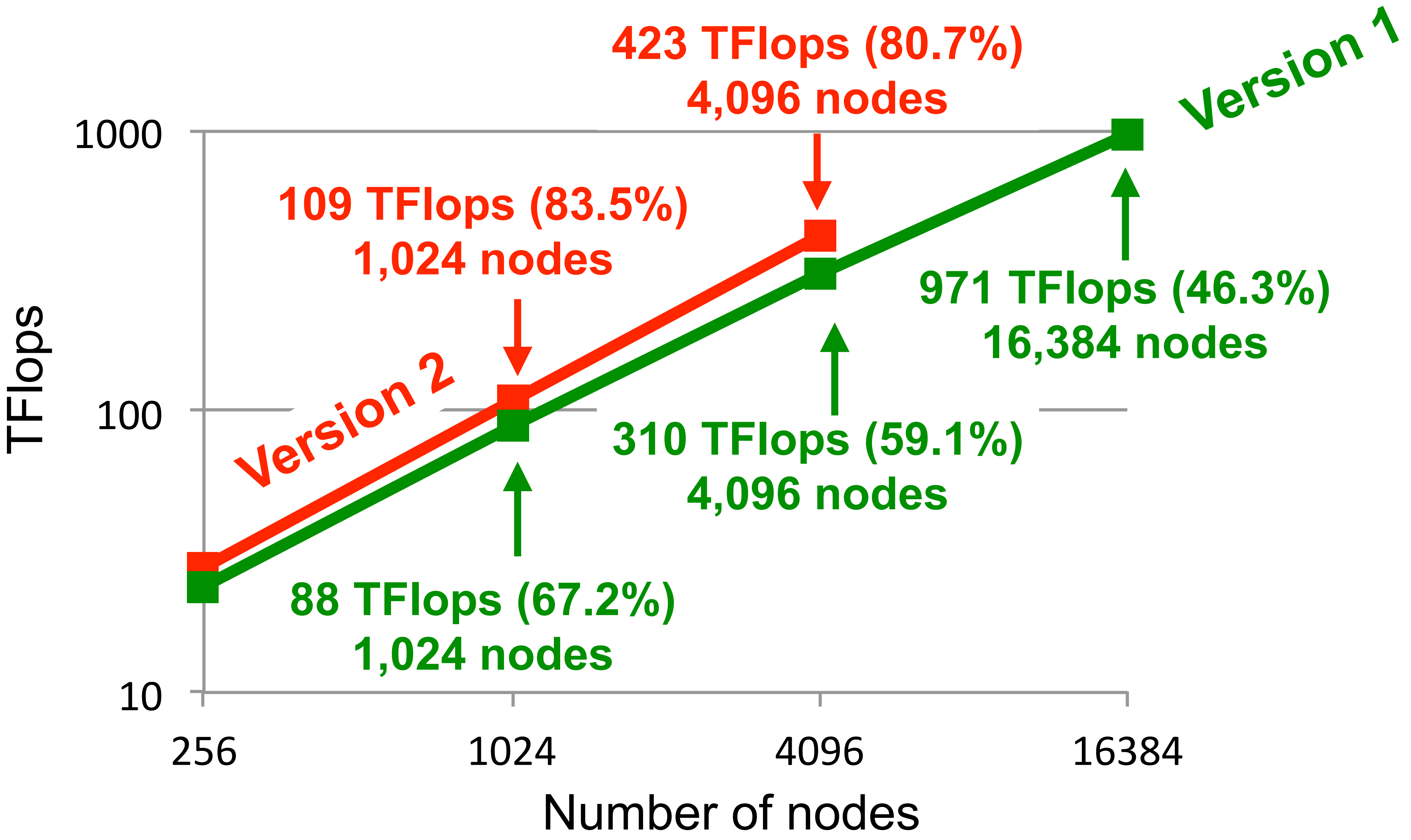
```
double A_L[N][NB];
#pragma xmp align A_L[i][*] with t(*,i)
    :
#pragma xmp gmove async(1)
A_L[k:len][0:NB] = A[k:len][j:NB];
    :
for(m=j+NB;m<N;m+=NB){
  for(n=j+NB;n<N;n+=NB){
    cblas_dgemm(&A[m][n], ..);
    if(xmp_test_async(1)){
       // receive A[k:len][j:NB];
           :
```



asynchronous broadcast communication

Confirm whether data with async clause comes or not.

Performance of HPL

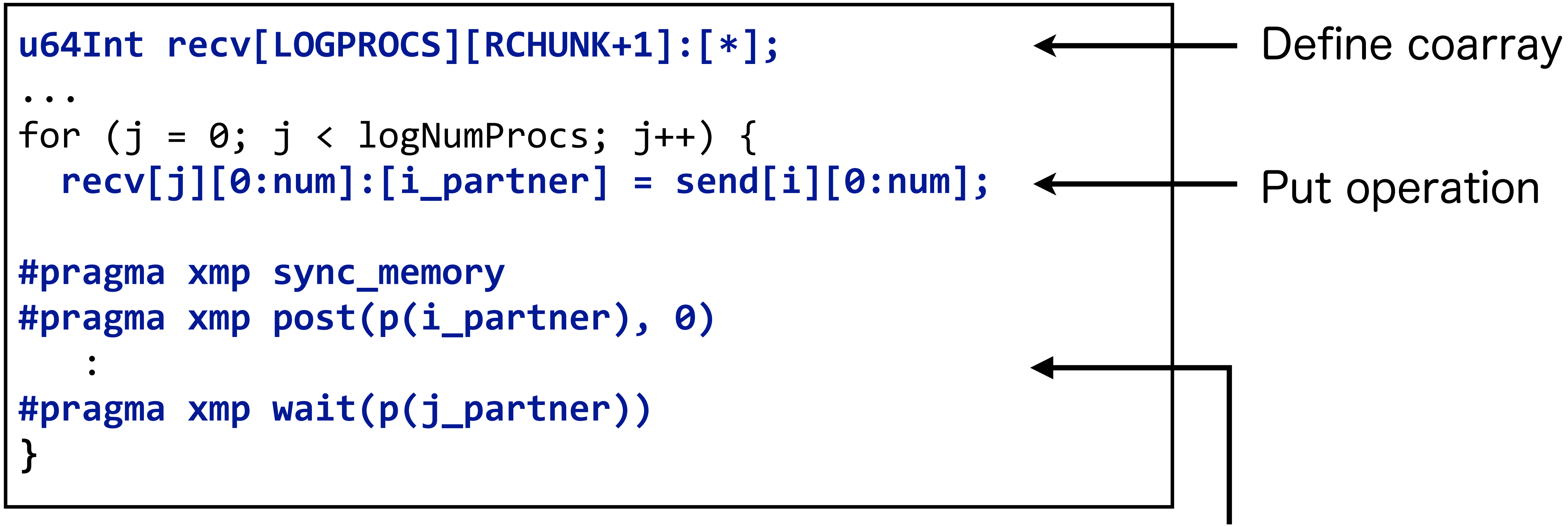


XMP-HPL Version 2 has a good scalability.

Sorry, the measurement in 16,384 nodes is late for this BoF.

RandomAccess

- SLOC is **253**, written in XMP/C
- Local-view programming with XMP/C coarray syntax
- The XMP RandomAccess is iterated over sets of CHUNK updates on each node



```
u64Int recv[LOGPROCS][RCHUNK+1]:[*];
...
for (j = 0; j < logNumProcs; j++) {
  recv[j][0:num]:[i_partner] = send[i][0:num];

#pragma xmp sync_memory
#pragma xmp post(p(i_partner), 0)
   :
#pragma xmp wait(p(j_partner))
}
```

Define coarray

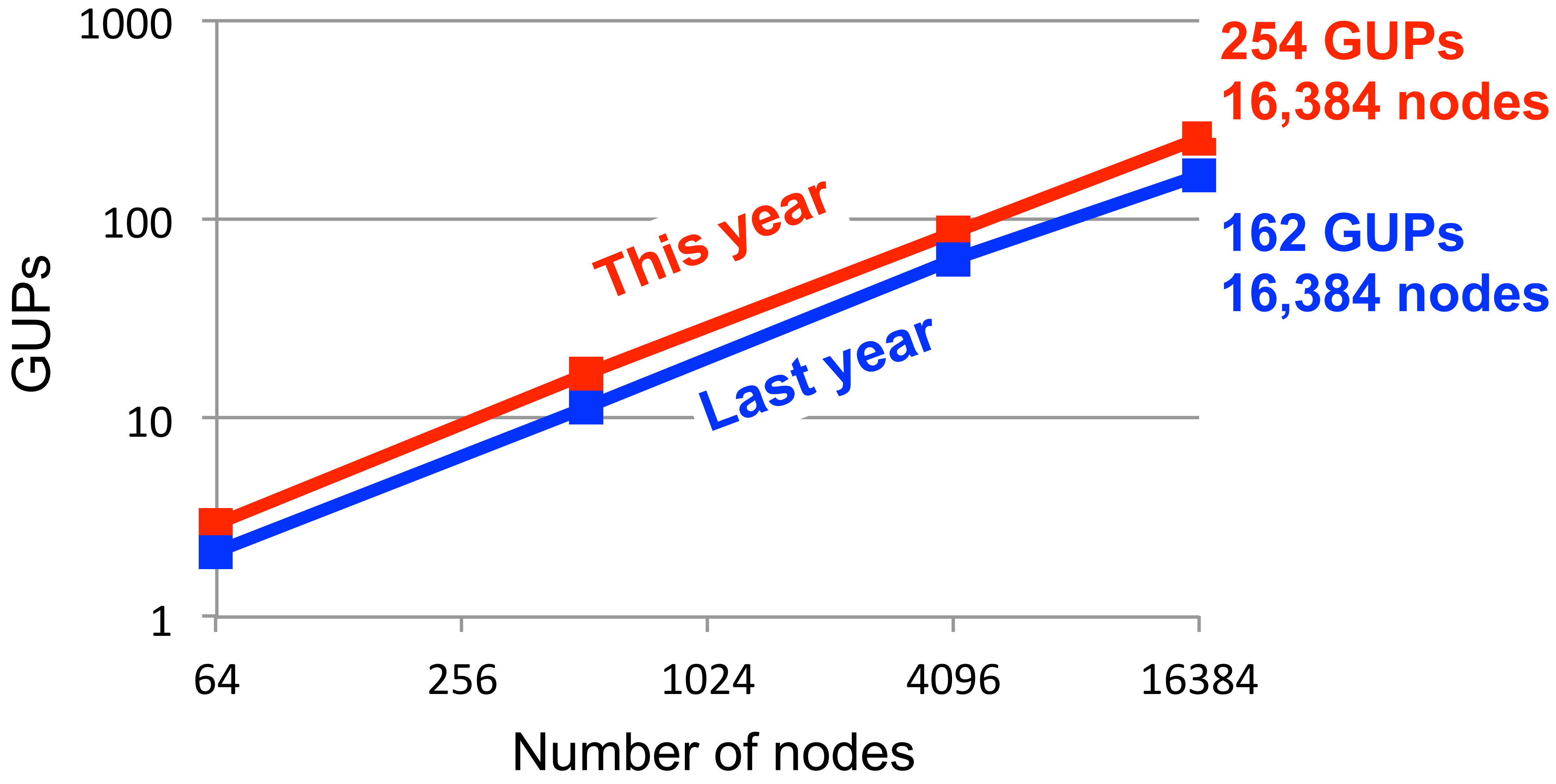
Put operation

A point-to-point synchronization is specified with the XMP's post and wait directives to realize asynchronous behavior of this algorithm

Performance of RandomAccess

Last year, to implement the post/wait directives, XMP uses MPI_Send/Recv.

This year, to implement them, XMP uses RDMA of the K computer.

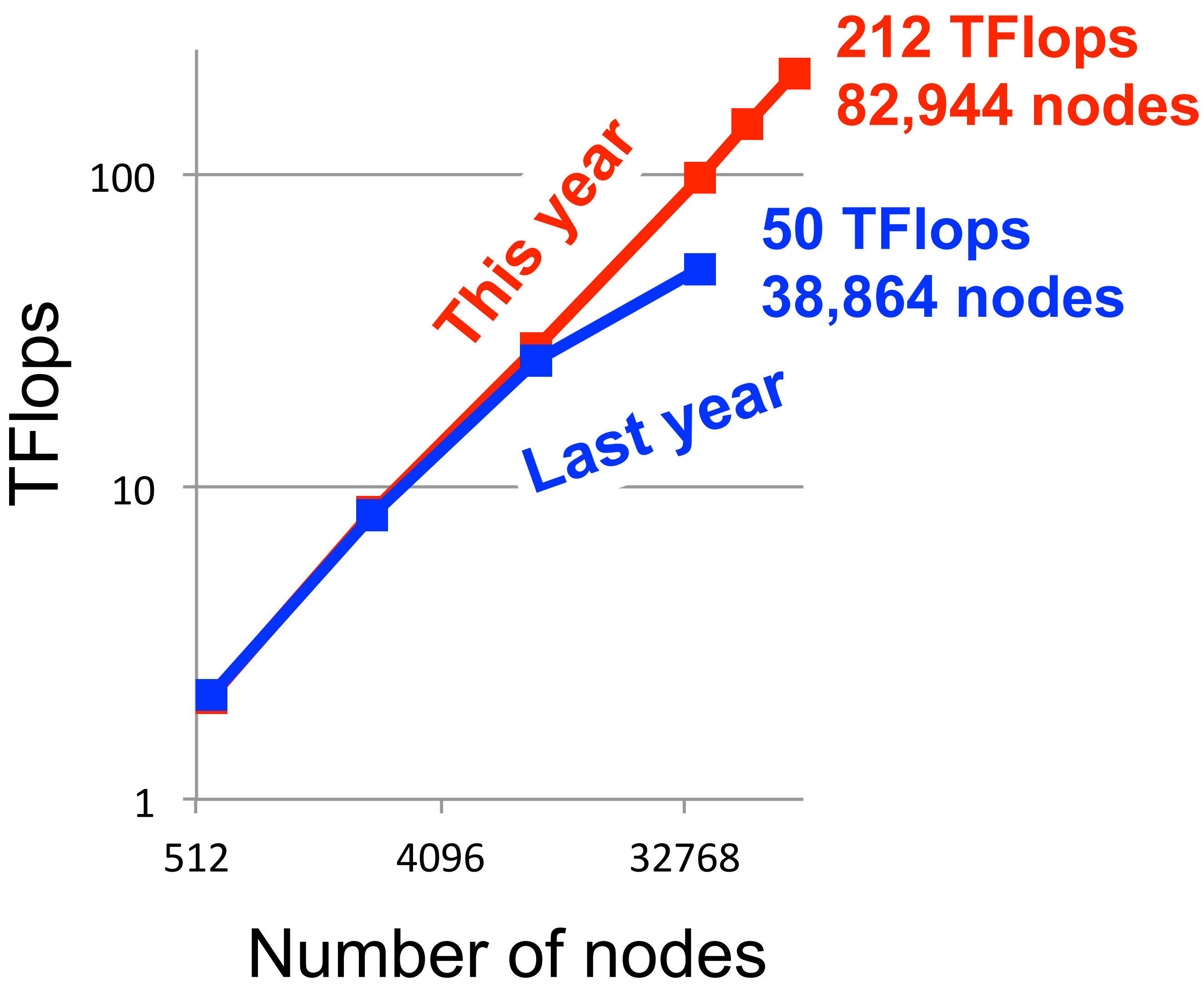


FFT and STREAM

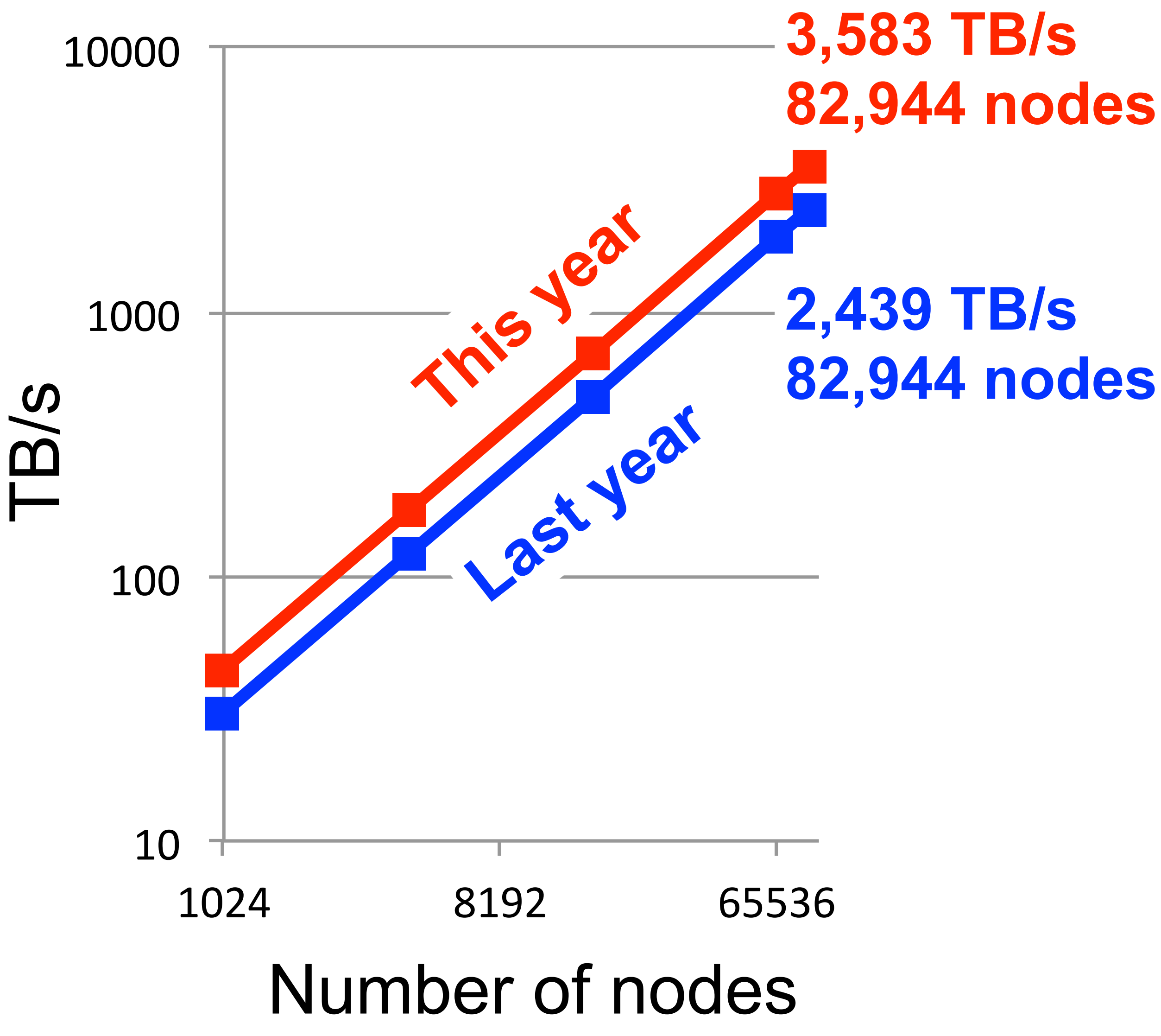
Code cleanup and performance improvement.

Please refer to the submission report at ***http://xcalablemp.org***

FFT (SLOC 205, XMP/F)

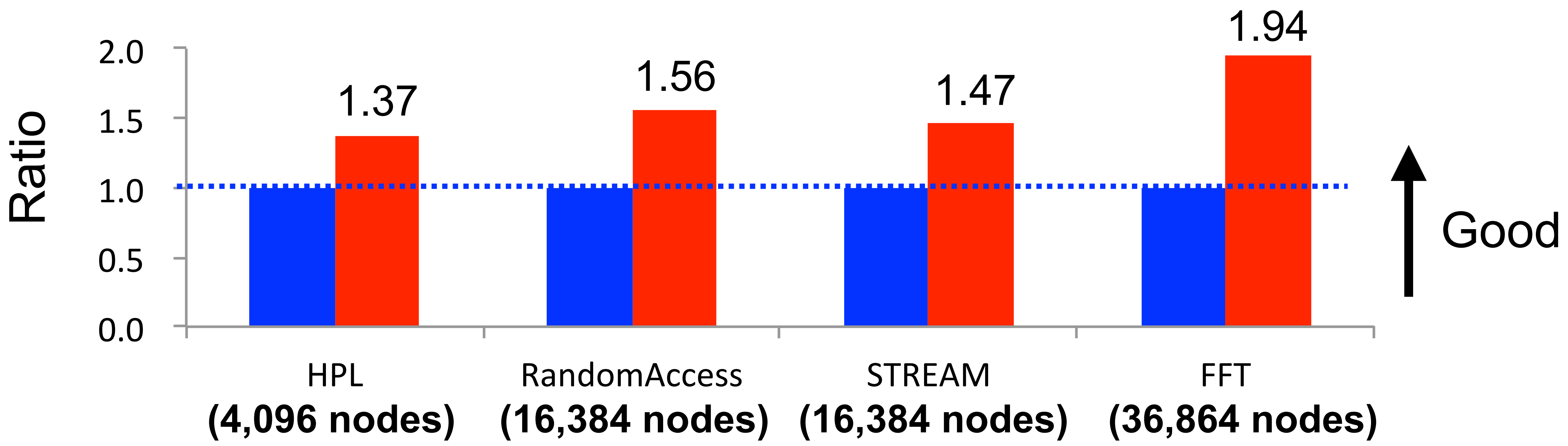


STREAM (SLOC 69, XMP/C)

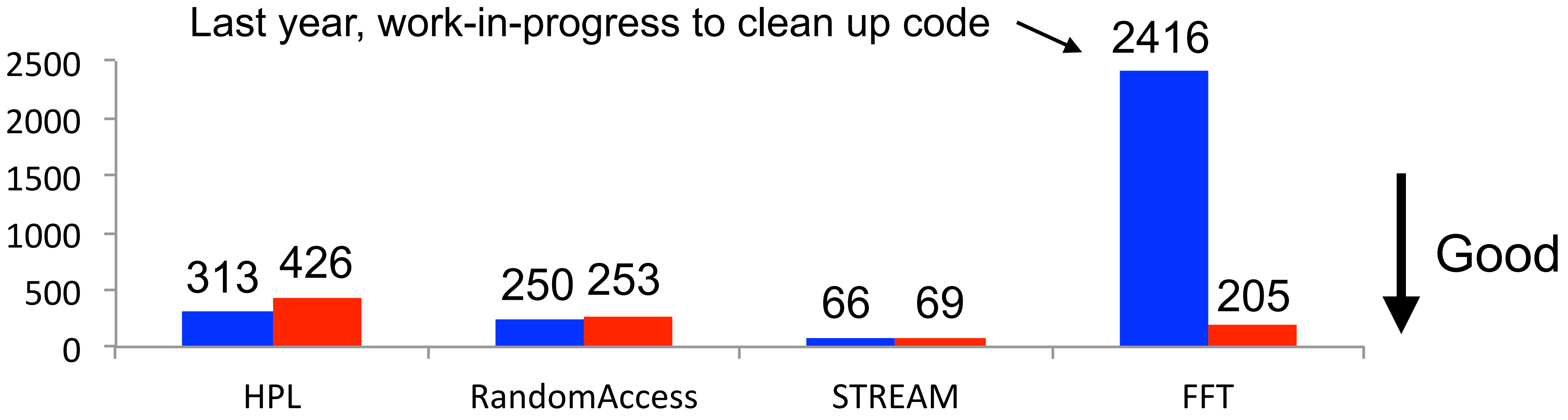


Compare to two versions

- Improvement rate (on the same nodes) **37 - 94% improvement !!**



- SLOC



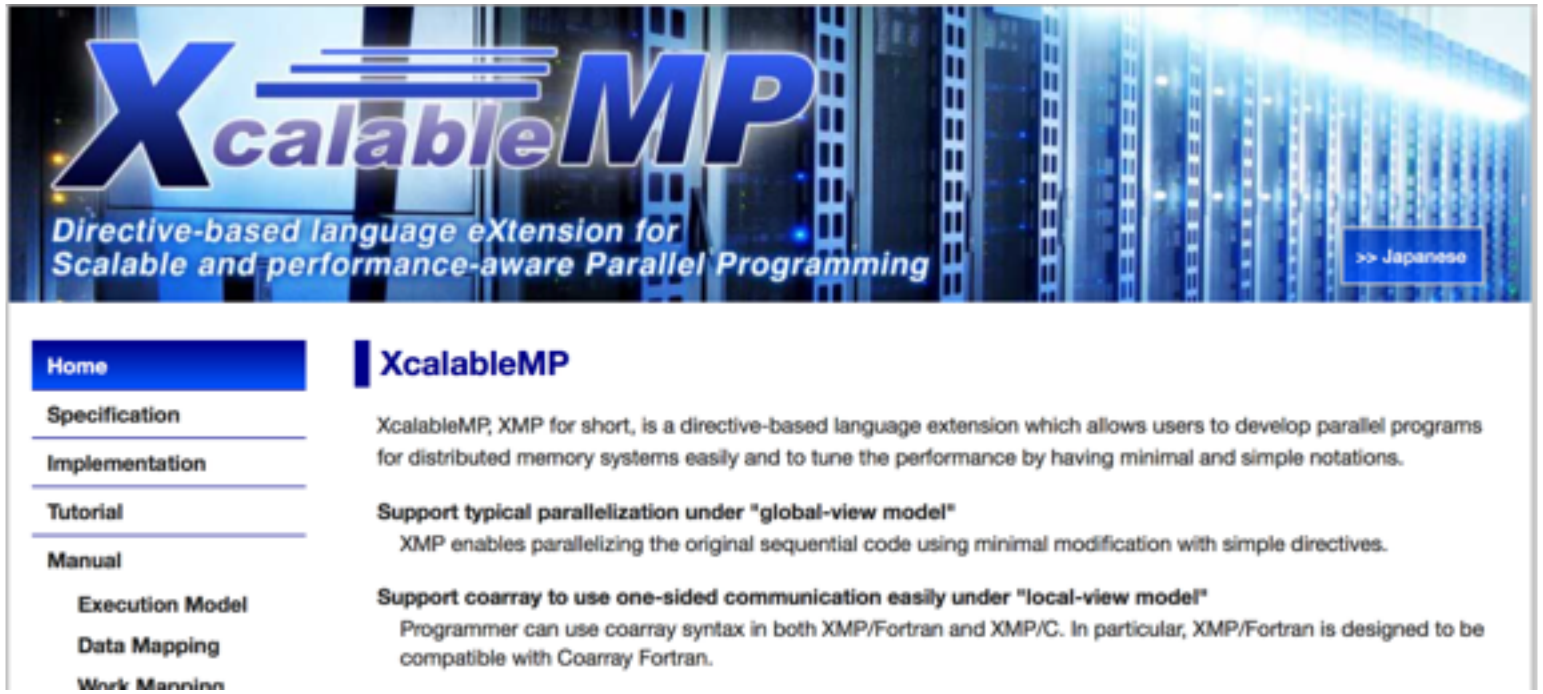
Outline

1. XcalableMP (XMP) for cluster systems (14min.)

2. XcalableACC (XACC) for accelerator cluster systems (6min.)
   Extension of XMP using OpenACC

Sorry !!, work-in-progress

The submission report is available at ***http://xcalablemp.org***



XcalableMP

XcalableMP, XMP for short, is a directive-based language extension which allows users to develop parallel programs for distributed memory systems easily and to tune the performance by having minimal and simple notations.

Support typical parallelization under "global-view model"
XMP enables parallelizing the original sequential code using minimal modification with simple directives.

Support coarray to use one-sided communication easily under "local-view model"
Programmer can use coarray syntax in both XMP/Fortran and XMP/C. In particular, XMP/Fortran is designed to be compatible with Coarray Fortran.

What is XcalableACC?

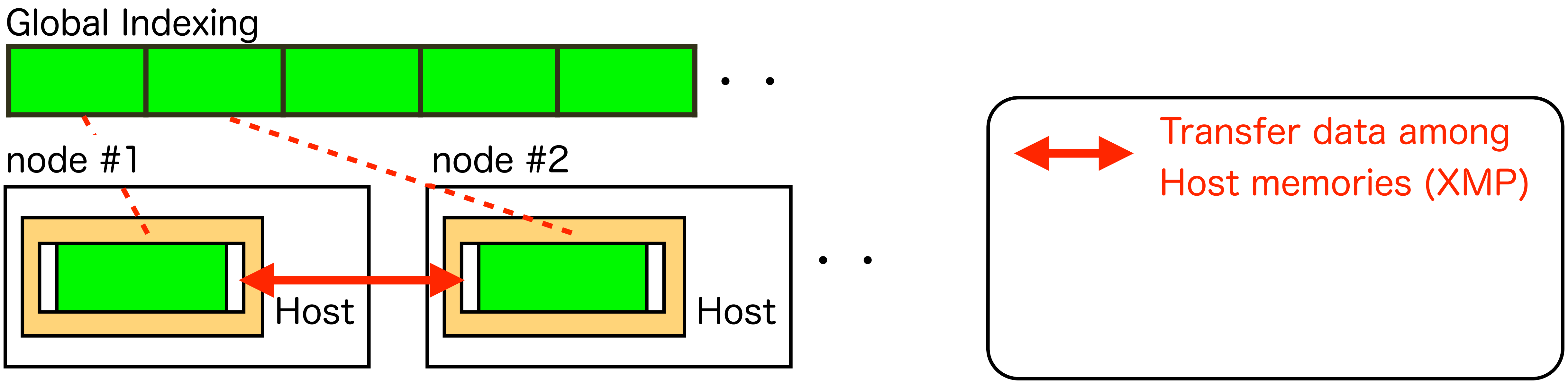
Extension of XMP using OpenACC for accelerator clusters

Feature:

- Mix XMP and OpenACC directives seamlessly
- Support transferring data among accelerators directly

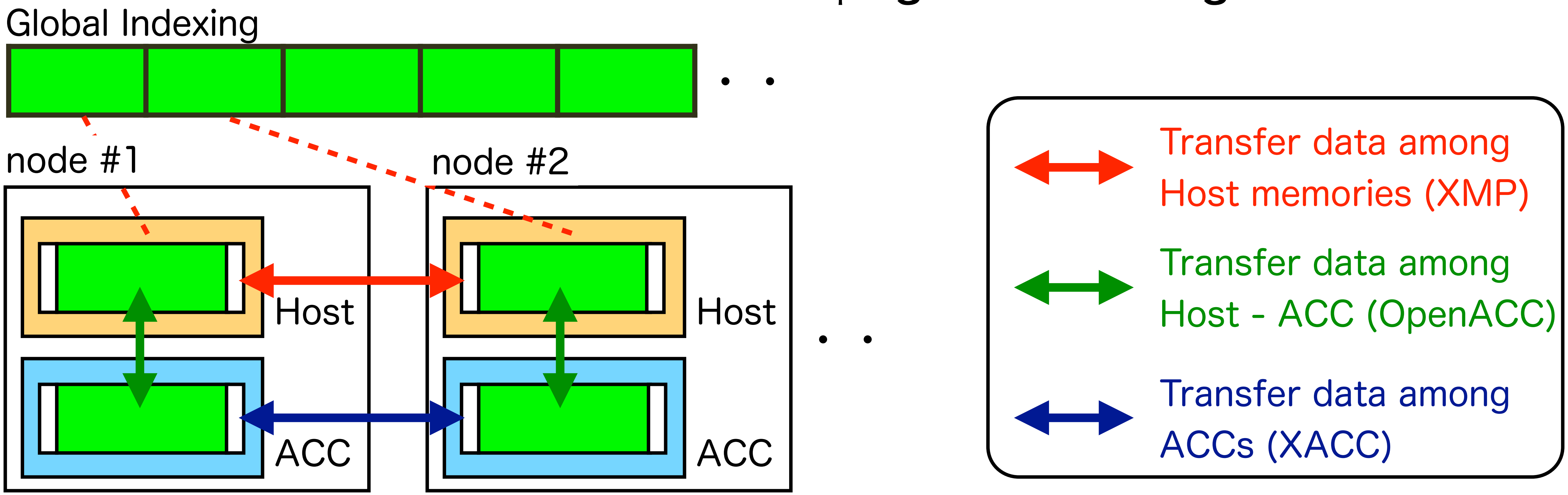
Difference XMP and XACC memory models

- **XMP** memory model



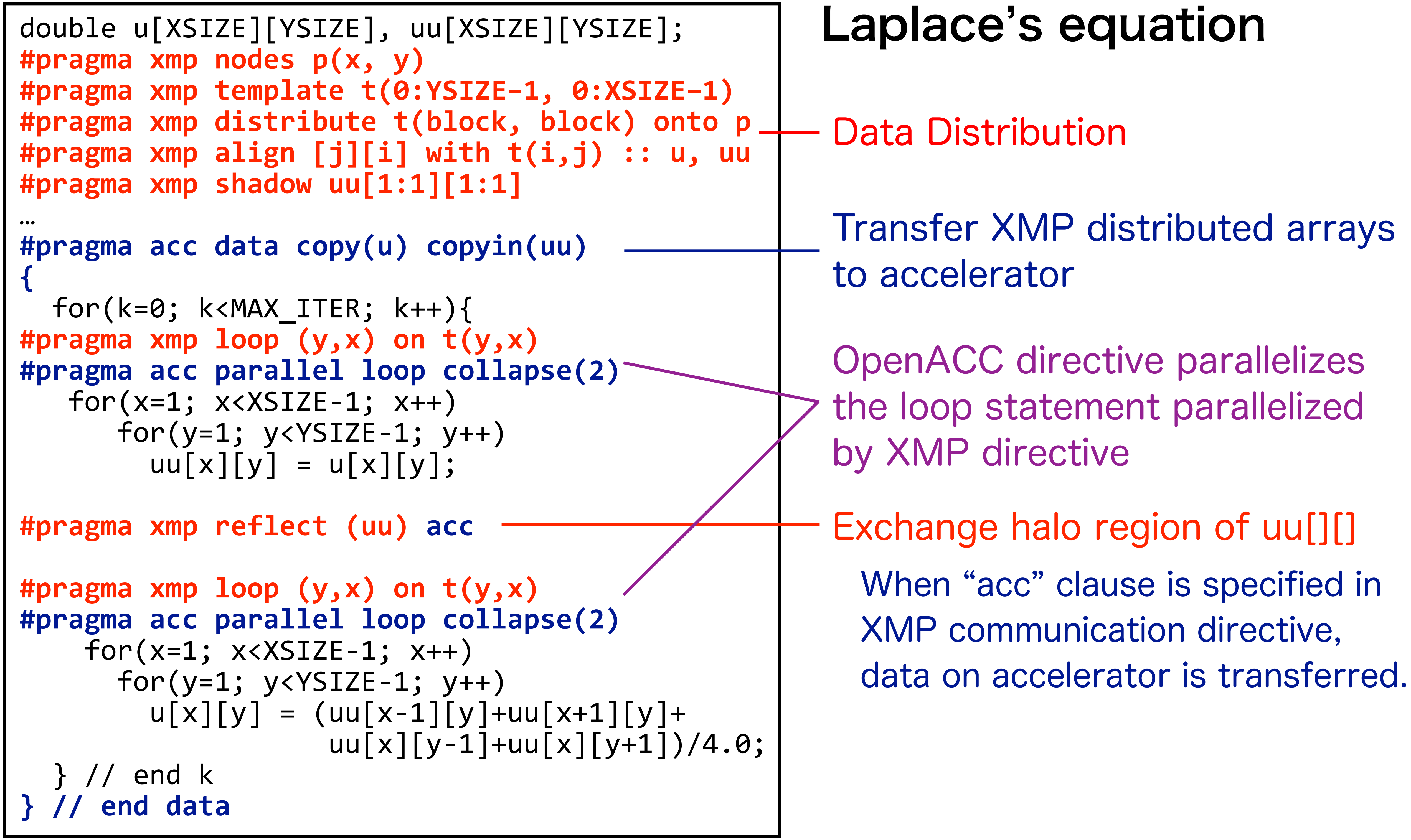
- **XACC** memory model

Map **“global Indexing”** to accelerators



XACC code example

Laplace’s equation



```
double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];
#pragma xmp nodes p(x, y)
#pragma xmp template t(0:YSIZE−1, 0:XSIZE−1)
#pragma xmp distribute t(block, block) onto p
#pragma xmp align [j][i] with t(i,j) :: u, uu
#pragma xmp shadow uu[1:1][1:1]
…
#pragma acc data copy(u) copyin(uu)
{
  for(k=0; k<MAX_ITER; k++){
#pragma xmp loop (y,x) on t(y,x)
#pragma acc parallel loop collapse(2)
   for(x=1; x<XSIZE-1; x++)
      for(y=1; y<YSIZE-1; y++)
        uu[x][y] = u[x][y];

#pragma xmp reflect (uu) acc

#pragma xmp loop (y,x) on t(y,x)
#pragma acc parallel loop collapse(2)
    for(x=1; x<XSIZE-1; x++)
      for(y=1; y<YSIZE-1; y++)
        u[x][y] = (uu[x-1][y]+uu[x+1][y]+
                   uu[x][y-1]+uu[x][y+1])/4.0;
  } // end k
} // end data
```

Data Distribution

Transfer XMP distributed arrays to accelerator

OpenACC directive parallelizes the loop statement parallelized by XMP directive

Exchange halo region of uu[][]

When “acc” clause is specified in XMP communication directive, data on accelerator is transferred.

Results and Machine

Summary

Three HPCC Benchmarks and HIMENO Benchmark

Benchmark	#Nodes	#CPUs	#GPUs	Performance (/peak)	SLOC
HPL	32	64	128	7 TFlops (4.2%)	343
FFT	32	64	-	257 GFlops (0.1%)	205
STREAM	64	128	256	15 TB/s (20.4%)	84
HIMENO	64	128	256	14 TFlops (1.4%)	253

HA-PACS/TCA: 64 nodes

- Ivy Bridge E5-2680v2, 224GFlops x 2 Sockets
- DDR3 SDRAM 128GB, 59.7GB/s x 2
- Infiniband 4xQDR x 2 rails : 8GB/s
- NVIDIA K20X (4GPUs / Node)
  - 1.31 TFlops/GPU(SP), 3.95 TFlops/GPU(DP)
  - 250GB/s/GPU

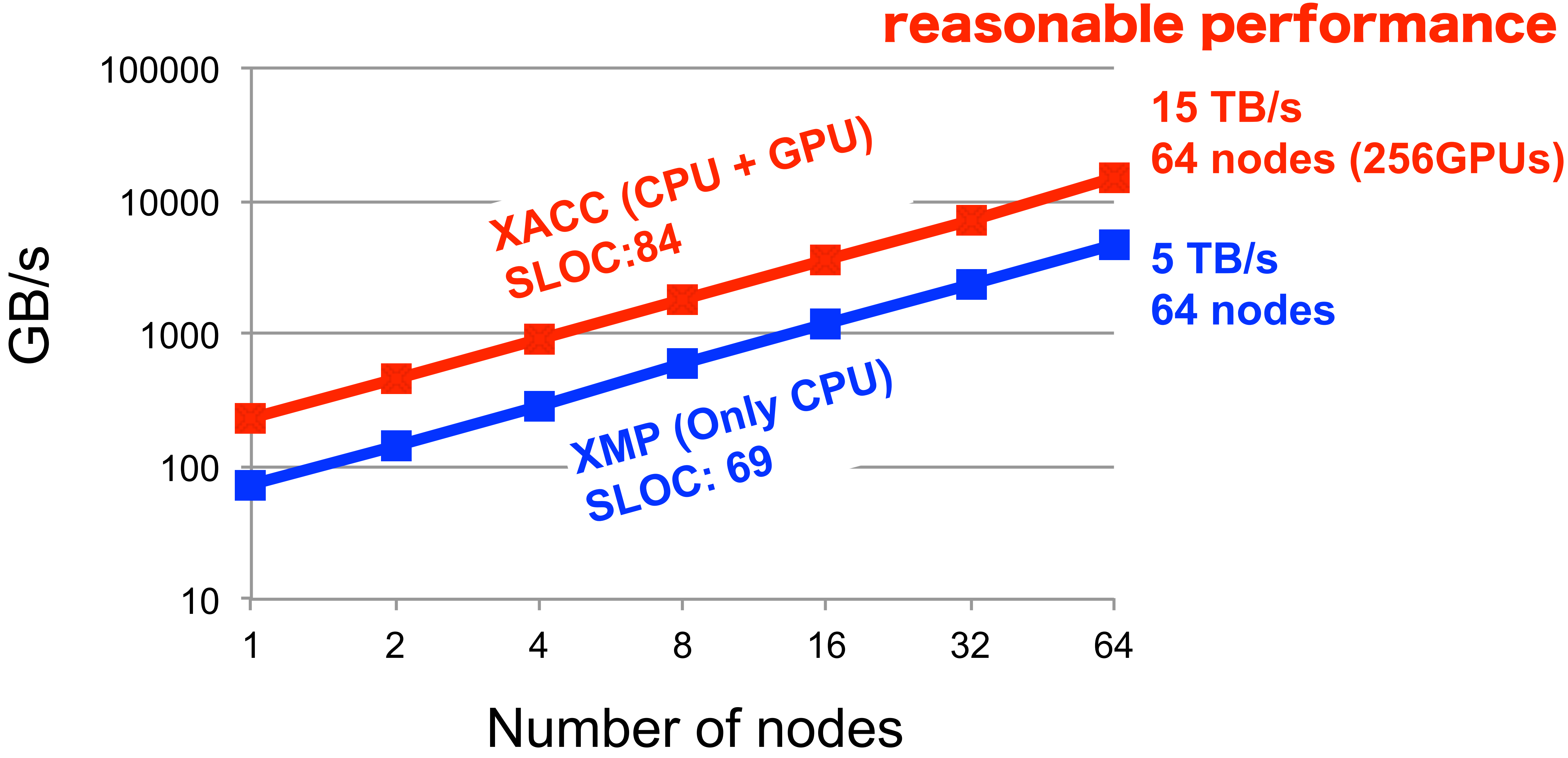
http://www.ccs.tsukuba.ac.jp/CCS/eng/research-activities/projects/ha-pacs

STREAM

The XACC STREAM uses both CPUs and GPUs together,
XMP, OpenACC, and **OpenMP** directives are used.

```
#pragma xmp nodes p(*)
#pragma acc data copy(a[0:GPU_SIZE], b[0:GPU_SIZE], c[0:GPU_SIZE])
 {
   for(k=0; k<NTIMES; k++) {
#pragma xmp barrier
   times[k] = -xmp_wtime();

#pragma acc parallel loop async
      for (j=0; j<GPU_SIZE; j++)                on GPU
        a[j] = b[j] + scalar*c[j];

#pragma omp parallel for
      for (j=GPU_SIZE; j<MAX_SIZE; j++)         on CPU
        a[j] = b[j] + scalar*c[j];

#pragma acc wait                                Wait until GPU task completes

#pragma xmp barrier
   times[k] += xmp_wtime();
   }
} // acc data
```

Performance of STREAM



HIMENO Benchmark

- Stencil application of incompressible fluid analysis code
- Solving the Poisson's equation
- Sequential and MPI Version HIMENO Benchmark is available at

http://accc.riken.jp/2444.htm

```
float p[MIMAX][MJMAX][MKMAX];
// Define distributed array and halo

#pragma acc data copy(p) ..
{
..
#pragma xmp reflect (p) acc
..
#pragma xmp loop (k,j,i) on t(k,j,i)
#pragma acc parallel loop ..
for(i=1; i<MIMAX; ++i)
  for(j=1; j<MJMAX; ++j){
#pragma acc loop vector ..
    for(k=1; k<MKMAX; ++k){
      S0 = p[i+1][j][k] * ..;
```

Transfer distributed array to accelerator

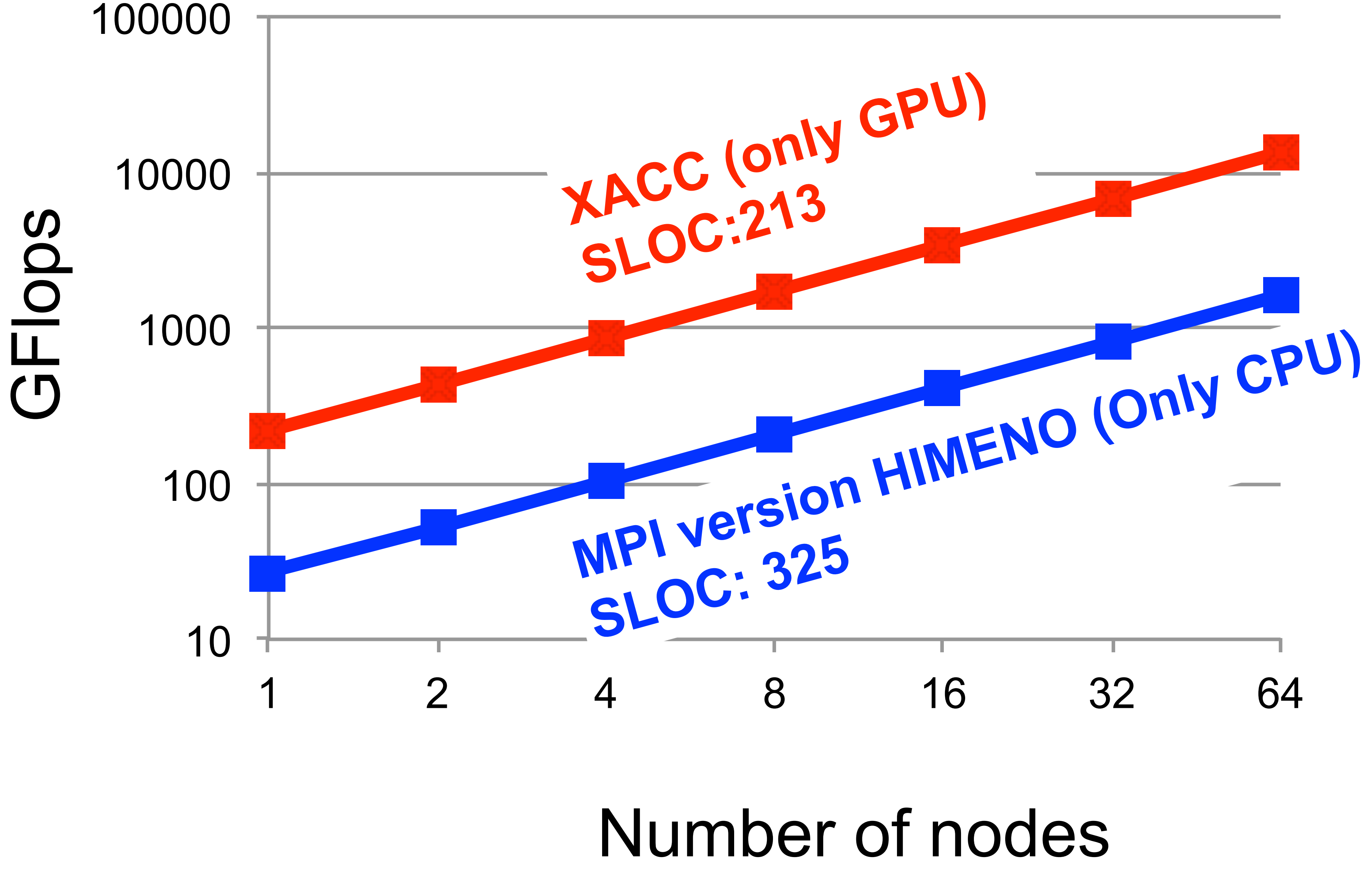
Exchange halo region

Parallelize loop statement

Only add XMP and OpenACC directives into the sequential Himeno benchmark.

Performance of HIMENO

reasonable performance



XACC (only GPU)
SLOC:213

MPI version HIMENO (Only CPU)
SLOC: 325

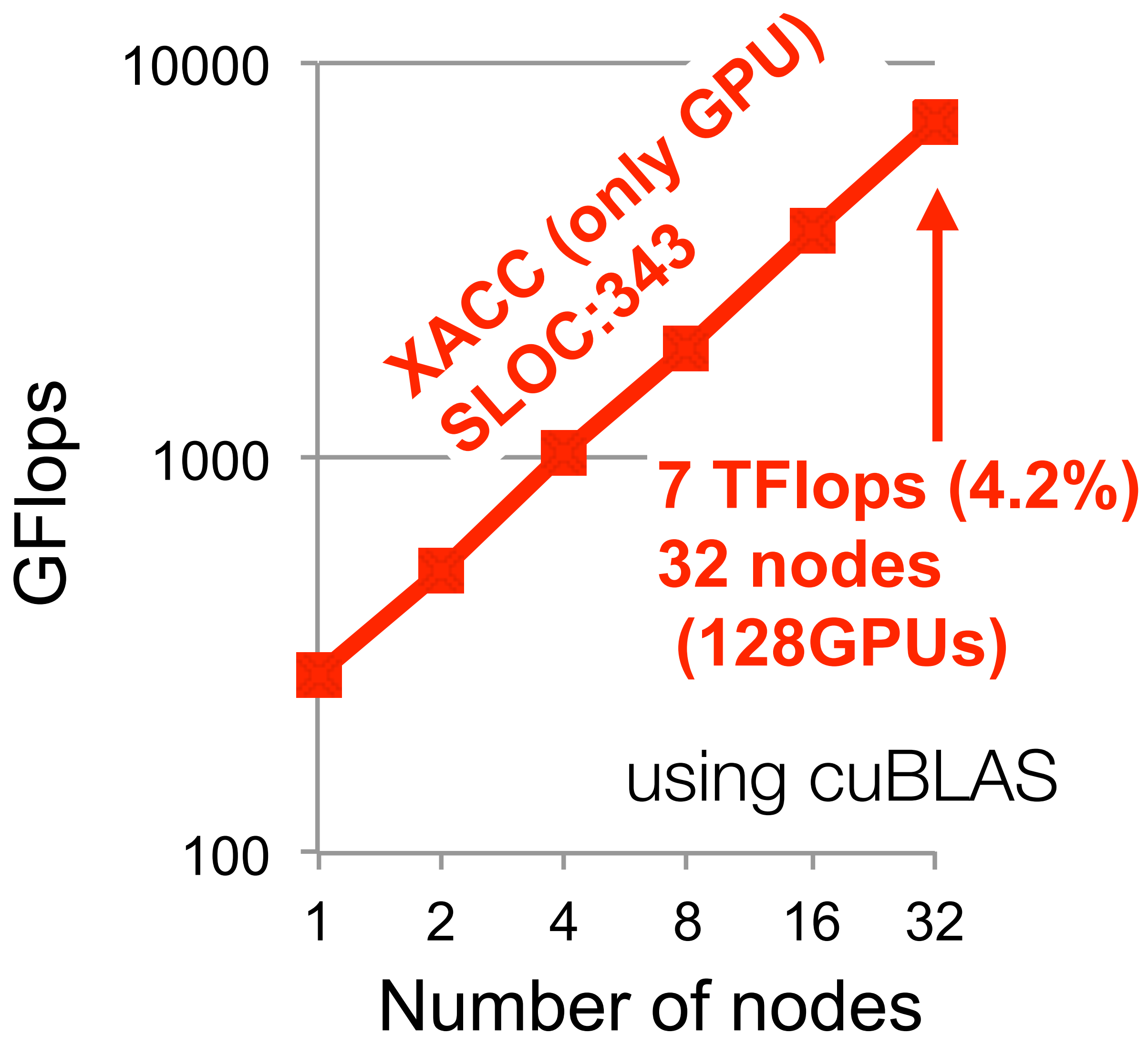
14 TFlops
64 nodes (256GPUs)

1.6 TFlops
64 nodes

HPL and FFT

Sorry !! work-in-progress for implementing and tuning.

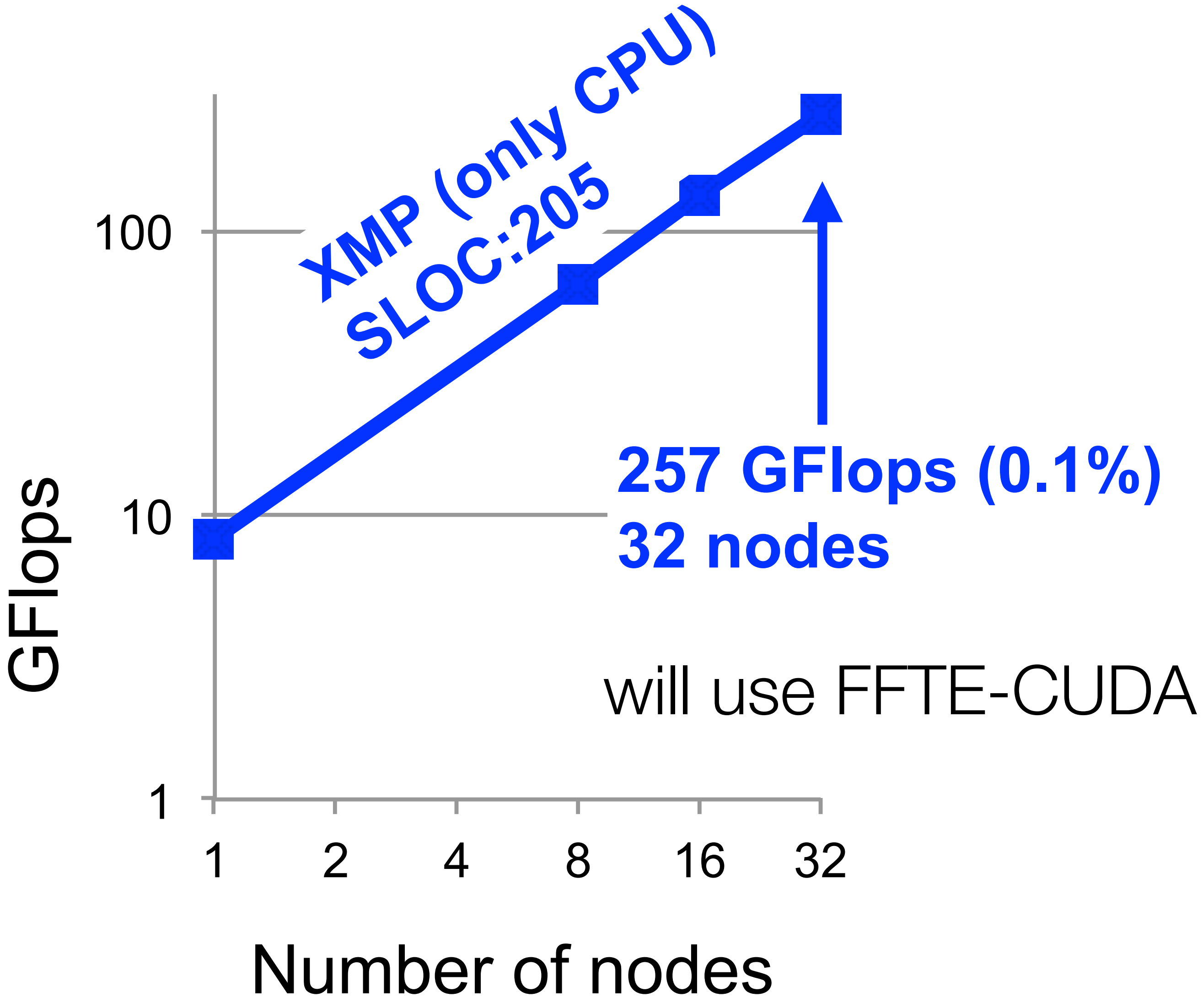
HPL



7 TFlops (4.2%)
32 nodes
(128GPUs)

using cuBLAS

FFT



257 GFlops (0.1%)
32 nodes

will use FFTE-CUDA

Time of transfer data between CPU and
host memory dominates the total computation time

Conclusion

XMP on the K computer

Good productivity and performance !!

Benchmark		# Nodes	Performance (/peak)	SLOC
HPL	Ver. 1	16,384	971 TFlops (46.3%)	313
	Ver. 2	4,096	423 TFlops (**80.7**%)	426
FFT		**82,944**	212 TFlops (2.0%)	205
STREAM		**82,994**	3,583 TB/s (67.5%)	69
RandomAccess		16,384	254 GUPs	253

XACC on HA-PACS/TCA

We will improve HPL and FFT next year.

Benchmark	#Nodes	#CPUs	#GPUs	Performance (/peak)	SLOC
HPL	32	64	128	7 TFlops (4.2%)	343
FFT	32	64	-	257 GFlops (0.1%)	205
STREAM	64	128	256	15 TB/s (20.4%)	84
HIMENO	64	128	256	14 TFlops (1.4%)	253

For more information

Please visit our booth !!

- RIKEN AICS (Advanced Institute for Computational Science) #2413
- Center for Computational Sciences, University of Tsukuba #3215