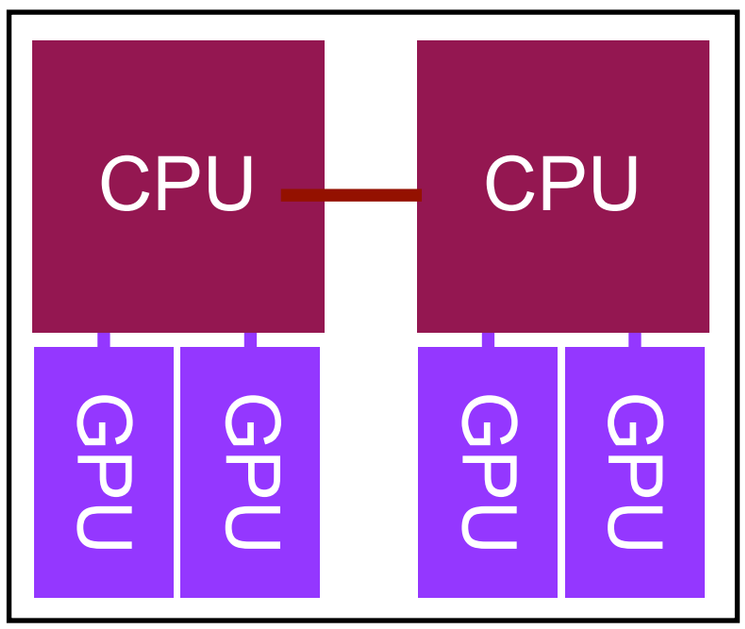
Evaluation of Productivity and Performance of the XcalableACC programming language

Masahiro Nakao (RIKEN AICS)

HA-PACS/TCA Cluster System

CPU	Intel Xeon-E5 2680v2 2.8 GHz x 2 Sockets
Memory	DDR3 SDRAM, 128GB, 59.7GB/s x 2
GPU	NVIDIA Tesla K20X x 4 GPUs, GDDR5 6GB x 4 and 250GB/s x 4*1
Interconnect	InfiniBand Mellanox Connect-X3 4 x QDR x 2rails 8GB/s

http://www.ccs.tsukuba.ac.jp



Each node has four GPUs (NVDIA K20X).
Therefore we assigned four processes to one node,
and each process deals with one GPU.

Objectives

- Evaluate Performance and Productivity of XcalableACC (XACC)
- Four benchmarks

HIMENO	Evaluate the performance of incompressible fluid analysis code (stencil code)
NPB CG	Solve minimum eigenvalue of symmetric and positive definite sparse matrix using the Conjugate Gradient method
STREAM	Evaluate sustainable memory bandwidth
HPL	High Performance Linpack. This code evaluates the floating point rate of execution for solving a linear system of equations

Objectives

- Evaluate Performance and Productivity of XcalableACC (XACC)
- Four benchmarks

HIMENO	Evaluate the performance of incompressible fluid analysis code (stencil code)
NPB CG	Solve minimum eigenvalue of symmetric and positive definite sparse matrix using the Conjugate Gradient method
STREAM	Evaluate sustainable memory bandwidth
HPL	High Performance Linpack. This code evaluates the floating point rate of execution for solving a linear system of equations

Implementation of HIMENO

```
float p[I][J][K];
#pragma xmp template t(0:K-1,0:J-1,0:I-1)
#pragma xmp nodes n(1, NDY, NDX)
#pragma xmp distribute t(block, block, ¥
                         block) onto n
#pragma xmp align p[k][j][i] with t(i, j, k)
#pragma xmp shadow p[1:2][1:2][0:1];

#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] * ..;
```

Define distributed array with halo region

Transfer distributed array to accelerator

Exchange halo region

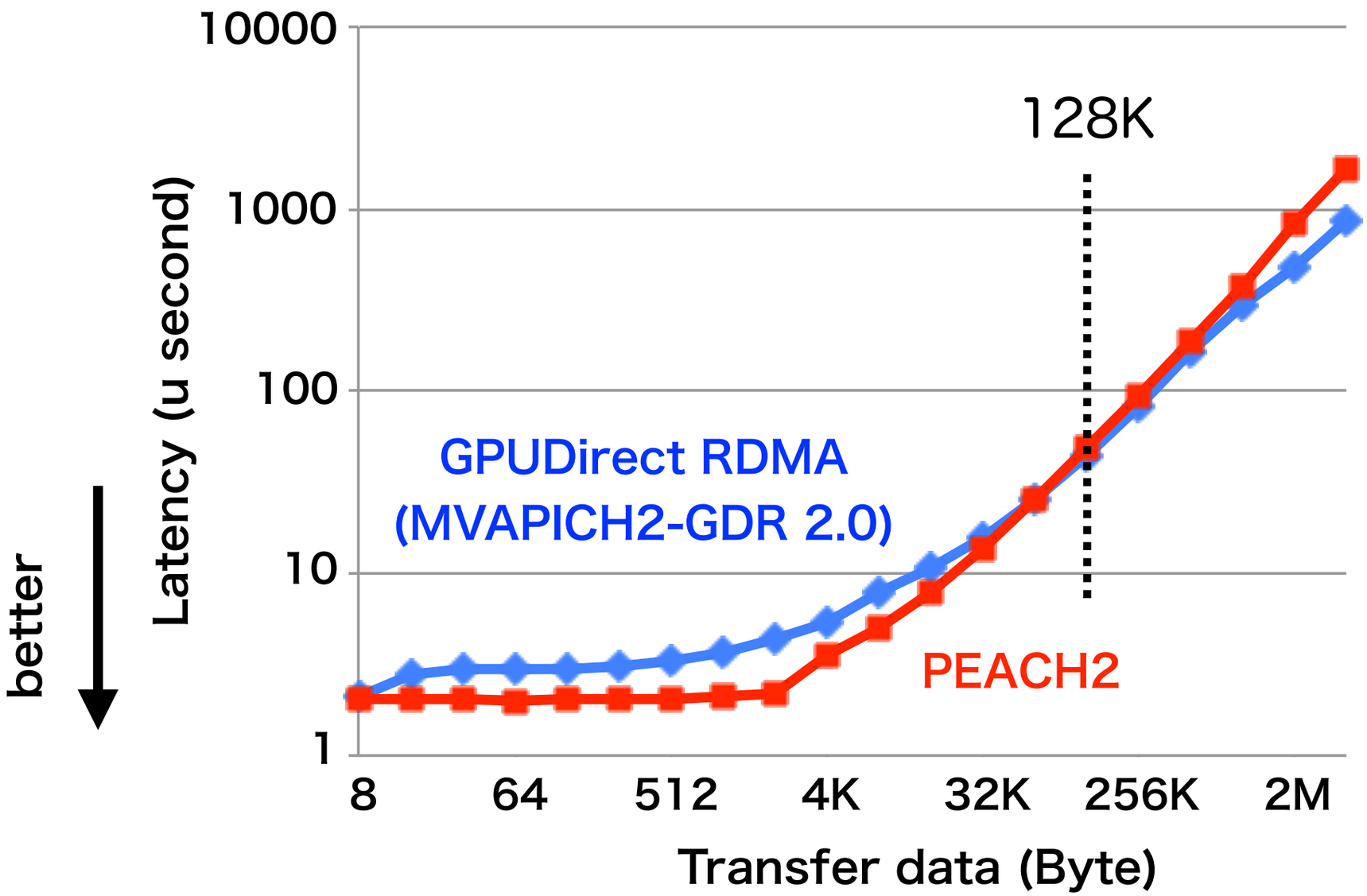
Parallelize loop statement

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

Pingpong on HA-PACS/TCA

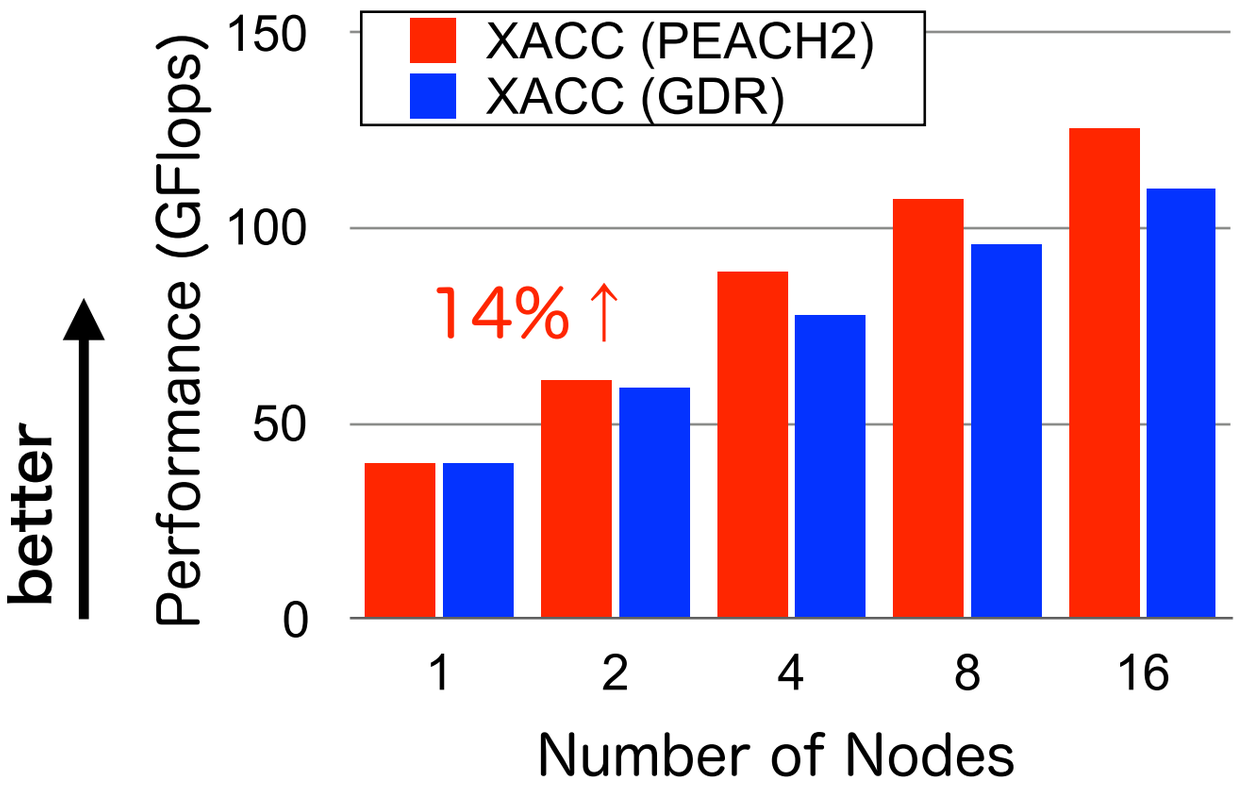
- PEACH2：**PCIe Gen.2 x 8links : 4GB/s**
- GPUDirect：**InfiniBand 4xQDR x 2rails : 8GB/s**

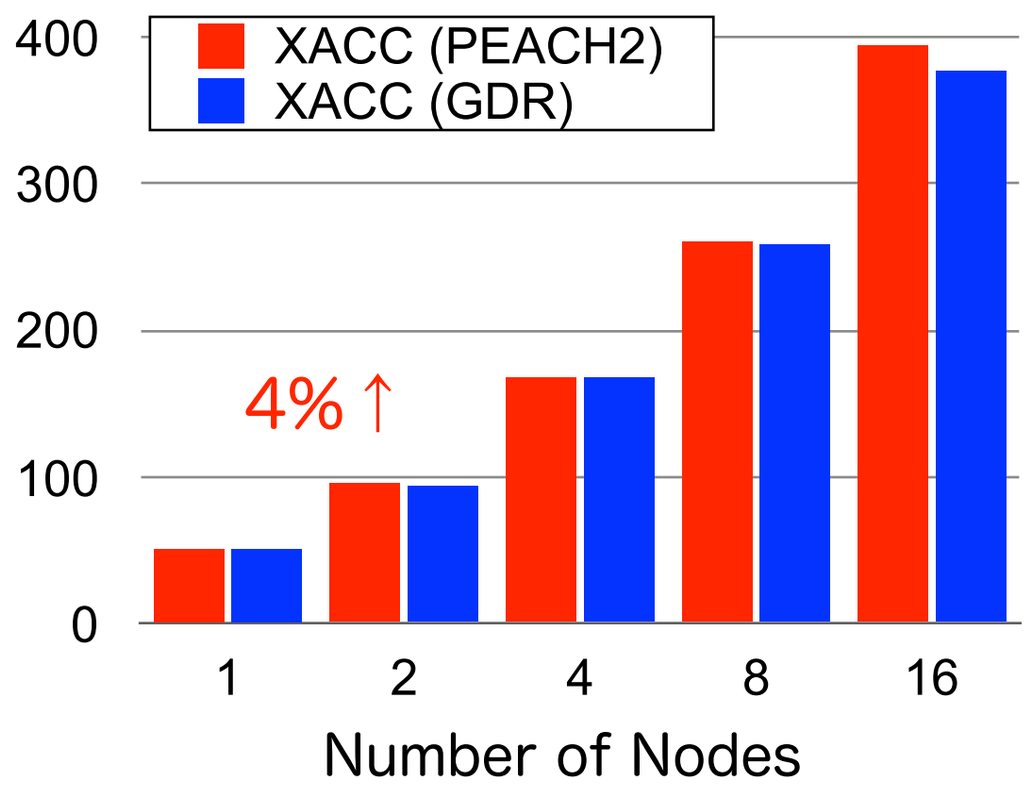
Device memory
to
Device memory
on neighbor nodes



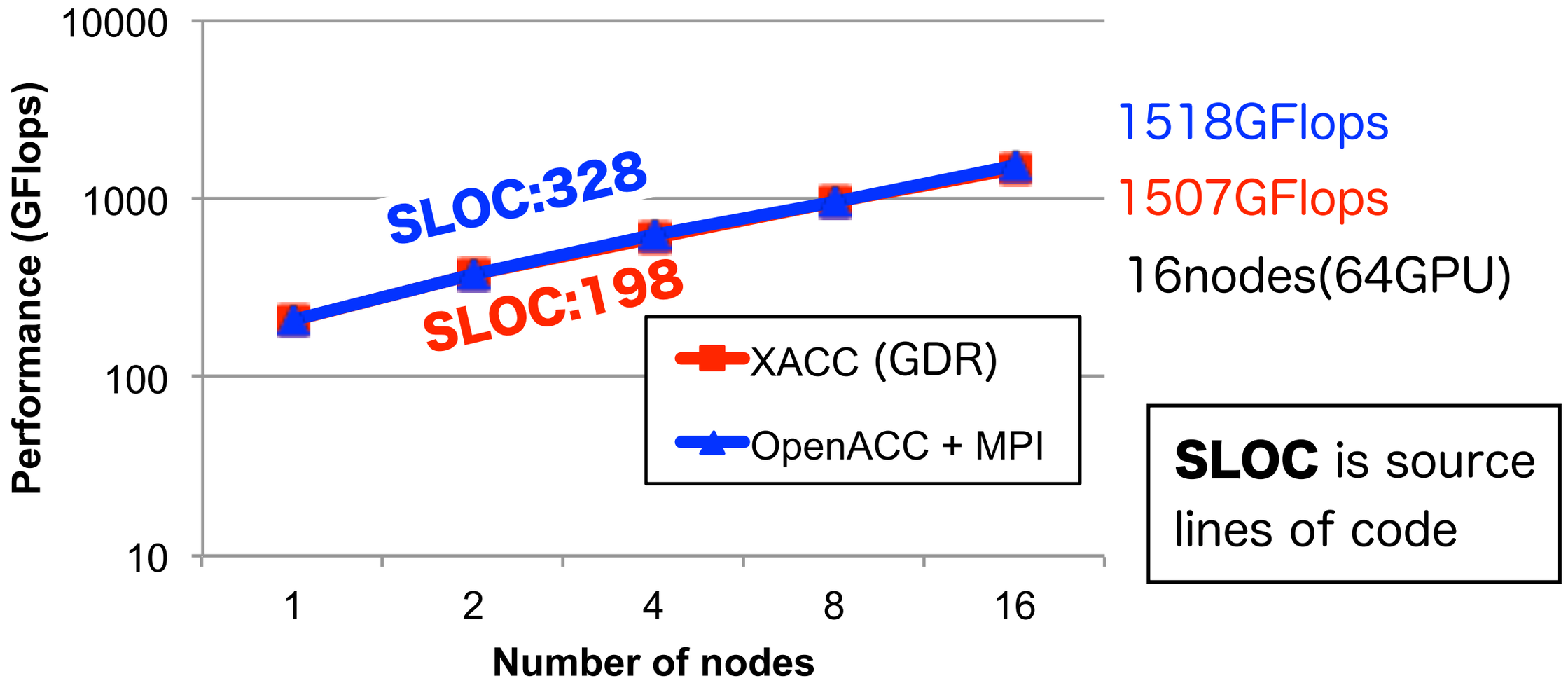
Performance of HIMENO (1/2)

- Comparison of “XACC with PEACH2” and “XACC with GDR (mvapich-GDR)”
- “XACC with PEACH2” is better than “XACC with GDR” in p[64][64][128]





Performance of HIMENO (2/2)



The performance of XACC is almost the same as that of OpenACC + MPI
SLOC of XACC is about 60% of that of OpenACC + MPI

Objectives

- Evaluate Performance and Productivity of XcalableACC (XACC)
- Four benchmarks

HIMENO	Evaluate the performance of incompressible fluid analysis code (stencil code)
NPB CG	Solve minimum eigenvalue of symmetric and positive definite sparse matrix using the Conjugate Gradient method
STREAM	Evaluate sustainable memory bandwidth
HPL	High Performance Linpack. This code evaluates the floating point rate of execution for solving a linear system of equations

Implementation of NPB CG

```
double w[NA];
#pragma xmp nodes p(PROC_COLS,PROC_ROWS)
#pragma xmp nodes sub_p(PROC_COLS)=p(:,*)
#pragma xmp template t(0:NA-1,0:NA-1)
#pragma xmp distribute t(block, block) onto p
#pragma xmp align w[i] with t(*,i)

for(cgit=1;cgit<=cgitmax;cgit++){
  rho0 = rho; d = 0.0; rho = 0.0;
#pragma xmp loop on t(*,j)
#pragma acc parallel loop gang
  for(j=0;j<NA;j++){
    double sum = 0.0;
    int rowstr_j = rowstr[j];
    int rowstr_j1 = rowstr[j+1];
#pragma acc loop vector reduction(+:sum)
    for(k=rowstr_j;k<rowstr_j1;k++){
      sum = sum + a[k]*p[colidx[k]];
    }
    w[j] = sum;
  } // for j
#pragma xmp reduction(+:w) on sub_p(:) acc
```

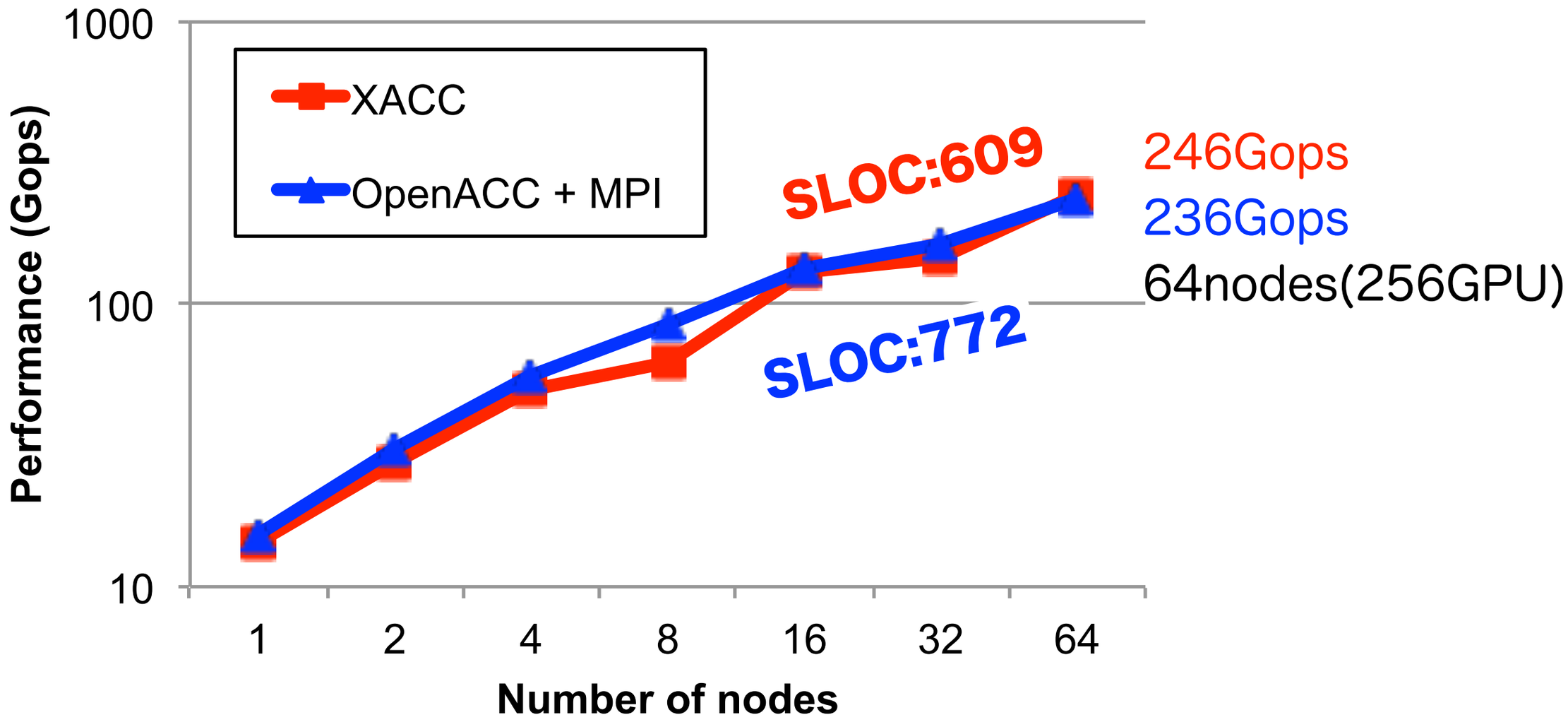
Define distributed array

Parallelize loop statement

Reduction on device memory

Reduction among nodes

Performance of NPB CG



The performance of XACC is almost the same as that of OpenACC + MPI.
SLOC of XACC is 79% of that of OpenACC + MPI.

Objectives

- Evaluate Performance and Productivity of XcalableACC (XACC)
- Four benchmarks

HIMENO	Evaluate the performance of incompressible fluid analysis code (stencil code)
NPB CG	Solve minimum eigenvalue of symmetric and positive definite sparse matrix using the Conjugate Gradient method
STREAM	Evaluate sustainable memory bandwidth
HPL	High Performance Linpack. This code evaluates the floating point rate of execution for solving a linear system of equations

Implementation of STREAM

- Evaluate sustainable memory bandwidth (a[i] = b[i] + scalar * c[i])

XMP

```
#pragma xmp nodes p(*)



#pragma xmp barrier
time = -xmp_wtime();



#pragma omp parallel for
for (i=0;i<N;i++)
  a[i] = b[i] + scalar*c[i];


#pragma xmp barrier
time += xmp_wtime();

GBs = calc_performance(time);
#pragma xmp reduction(+:GBs)
```

XACC

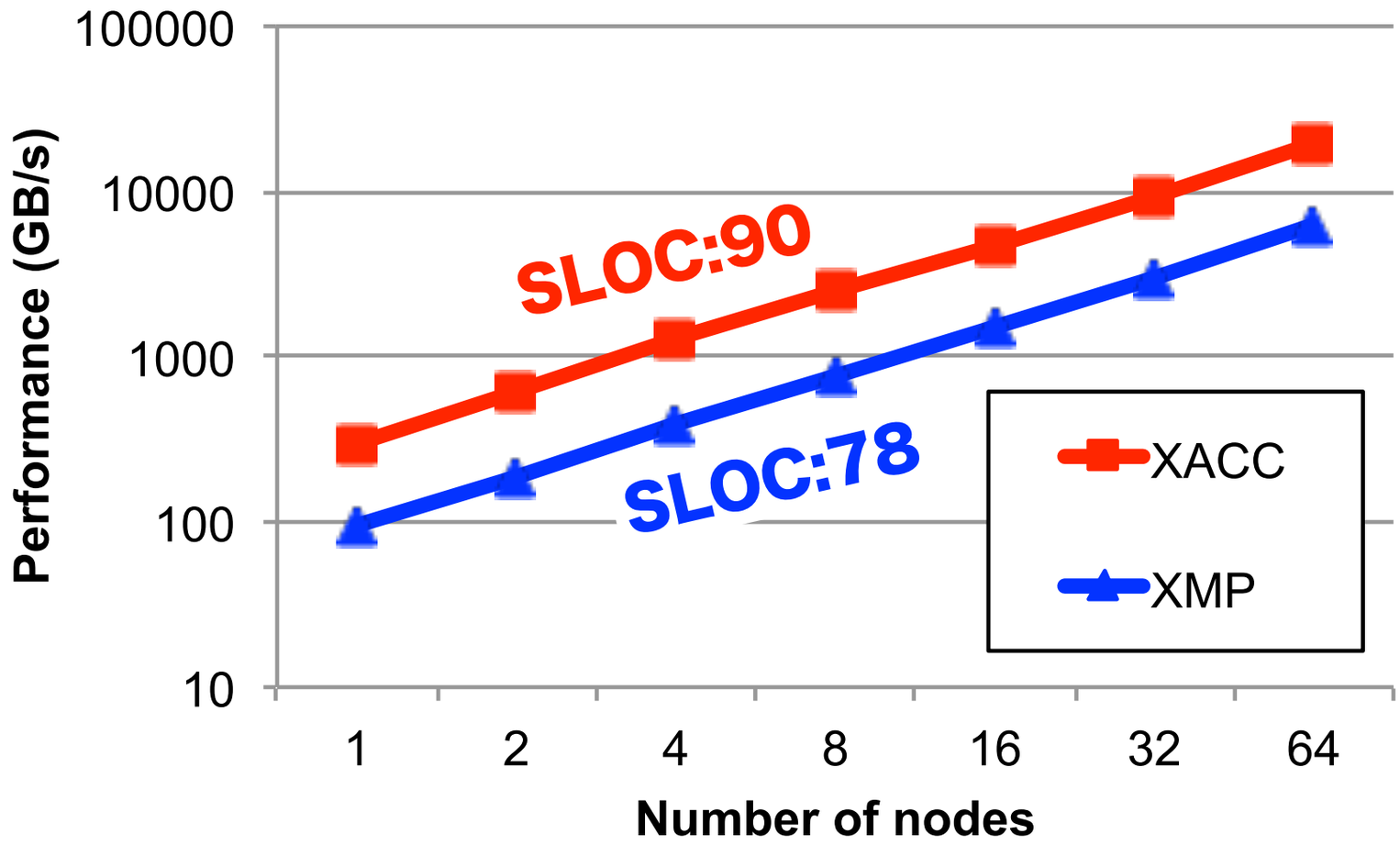
```
#pragma xmp nodes p(*)

#pragma acc data copy(a[:GSIZE], b[:GSIZE], c[:GSIZE])
{
#pragma xmp barrier
  time += xmp_wtime();

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

#pragma omp parallel for
  for(i=GSIZE;i<N;i++)
    a[i] = b[i] + scalar*c[i];

#pragma acc wait
#pragma xmp barrier
  time += xmp_wtime();
}
GBs = calc_performance(time);
#pragma xmp reduction(+:GBs)
```

Accelerator executes asynchronously

Host executes

Wait for completion of above accelerator execution.

Performance of STREAM



18,895GB/s

6,067GB/s

64nodes(256GPU)

The performance of XACC is 3.2 times better than that of XMP.
(Note that XACC uses both GPU and CPU, and XMP uses only CPU.)

Objectives

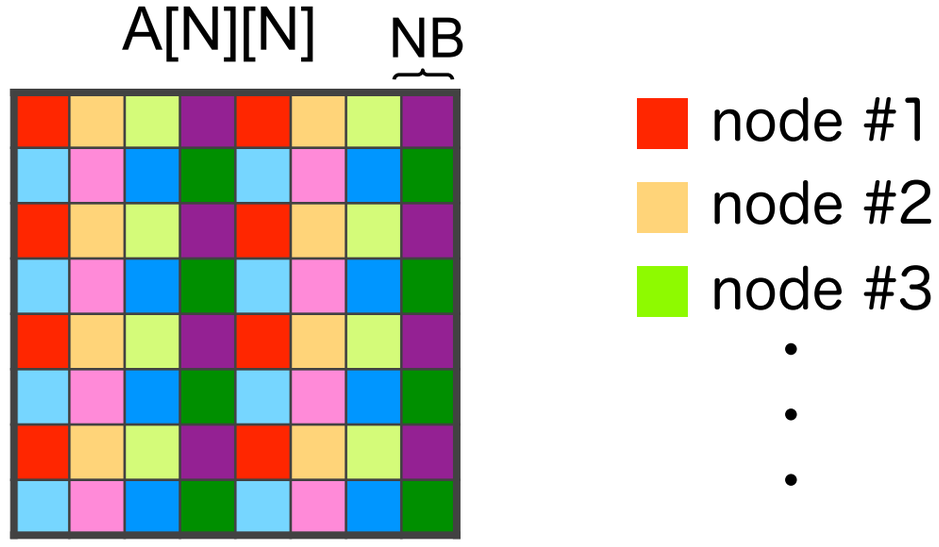
- Evaluate Performance and Productivity of XcalableACC (XACC)
- Four benchmarks

HIMENO	Evaluate the performance of incompressible fluid analysis code (stencil code)
NPB CG	Solve minimum eigenvalue of symmetric and positive definite sparse matrix using the Conjugate Gradient method
STREAM	Evaluate sustainable memory bandwidth
HPL	High Performance Linpack. This code evaluates the floating point rate of execution for solving a linear system of equations

Implementation of HPL

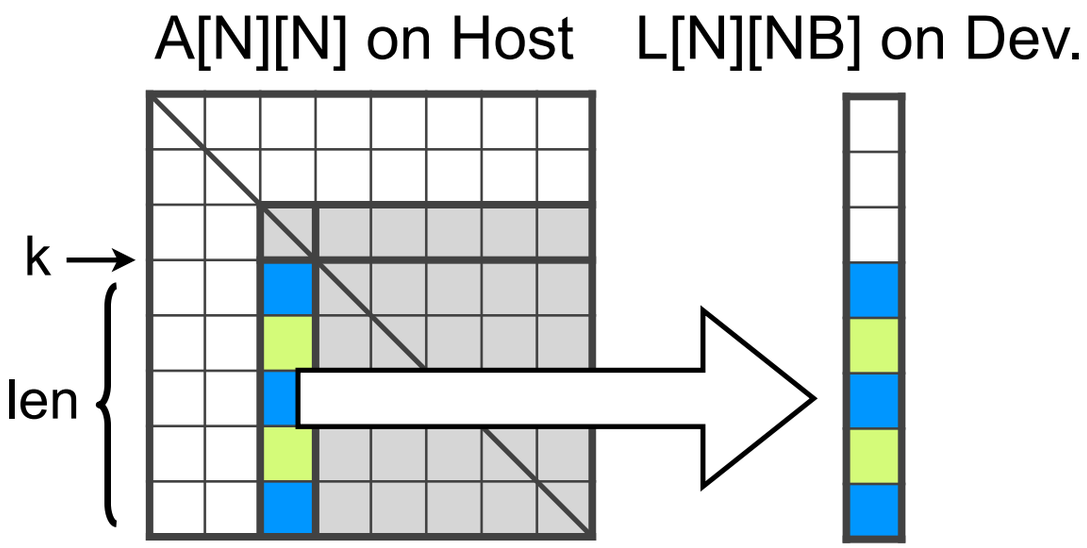
1. Block-cyclic distribution for coefficient matrix

```
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)
```



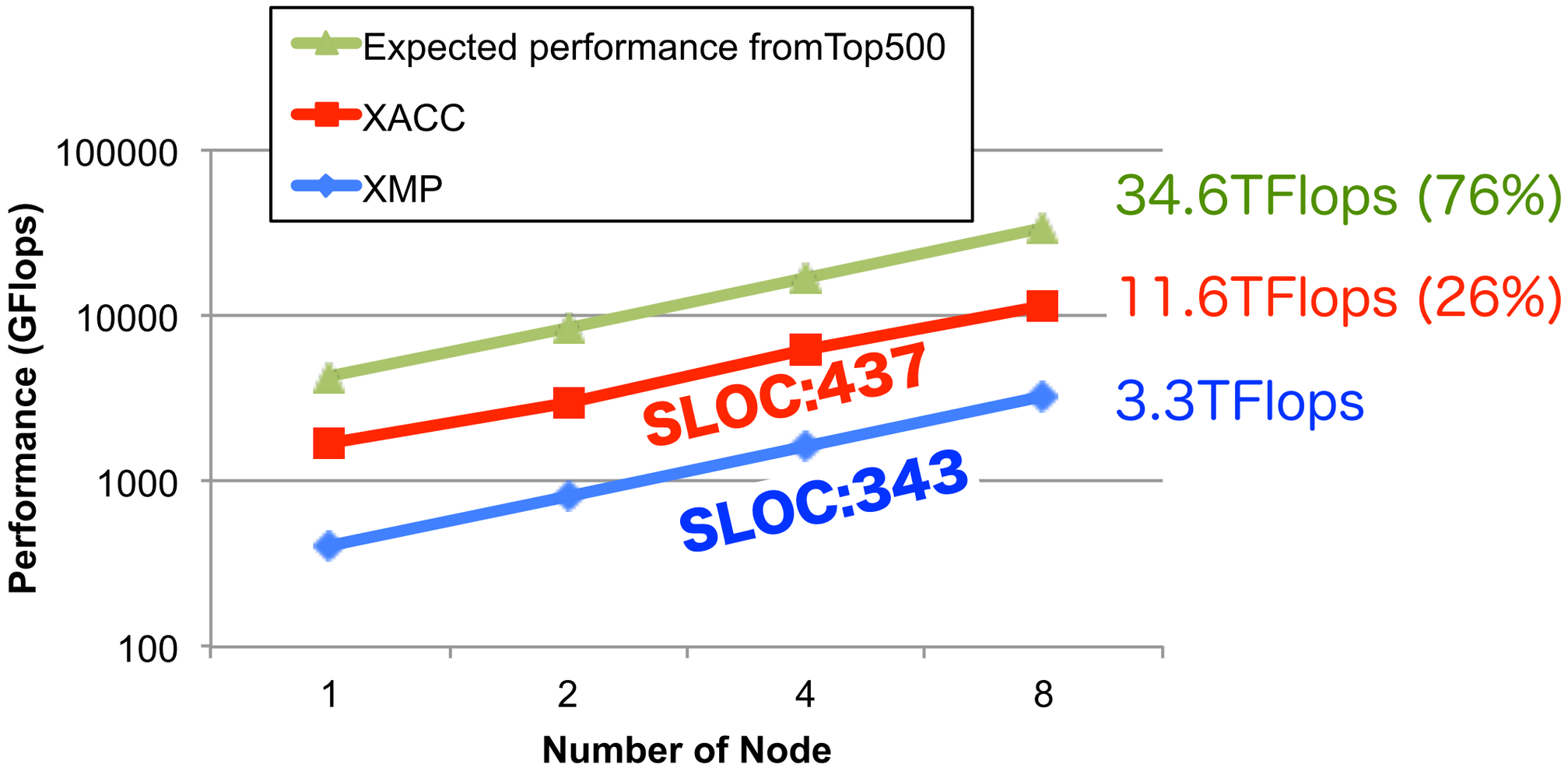
2. Panel Broadcast from host memory to device memory

```
double L[N][NB];
#pragma xmp align L[i][*] with t(*,i)
#pragma acc enter data create(L[:][:])
     :
#pragma xmp gmove acc(L)
L[k:len][0:NB] = A[k:len][k-NB:NB];
```



3. Update matrix : Use cuBLAS DGEMM developed by NVIDIA

Performance of HPL



The performance in Top500 is used by using CUDA + MPI version HPL developed by NVIDIA.
The DGEMM kernel is different ?? Under investigation.

Conclusion

- Objective

  Evaluation of productivity and performance on XACC

- Evaluation
  - In HIMENO, XACC using PEACH2 is better than that of mvapich-GDR in small data size
  - SLOCs of XACC is smaller than those of OpenACC + MPI, typical programing model
  - Performances of XACC is the almost the same as those of OpenACC + MPI except for HPL. Now we are tuning XACC version HPL.

- Future plan

  Real world application with N-body simulations in space scientific field (collaborate with Yohei Miki)