Performance Evaluation for Omni XcalableMP Compiler on Many-core Cluster System based on Knights Landing

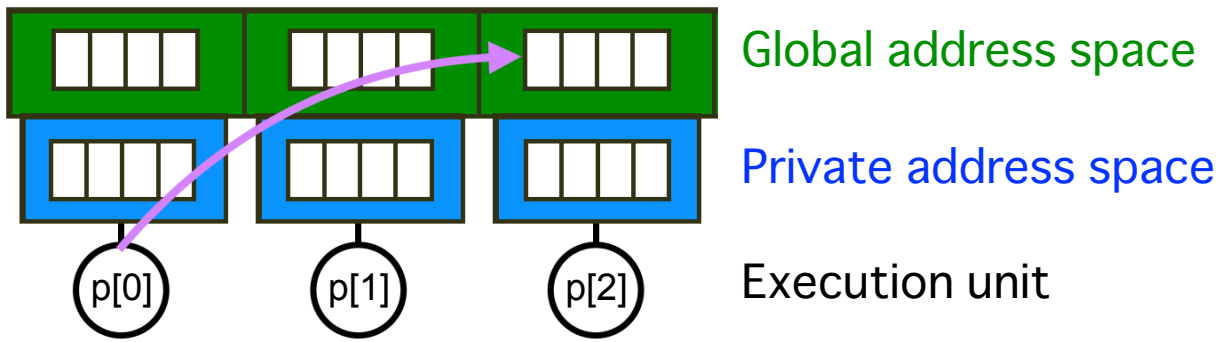
Masahiro Nakao[1], Hitoshi Murai[1], Taisuke Boku[2], Mitsuhisa Sato[1]

1. RIKEN Advanced Institute for Computational Science
2.Center for Computational Sciences University of Tsukuba

IXPUG Workshop Asia 2018, Jan. 31th, 2018@Tokyo, Japan

Background (1/2)

- Partitioned Global Address Space (PGAS) programming model for cluster system
  - Provide **global address space** on distributed memory system
  - Higher productivity then MPI
  - **XcalableMP (XMP)**, XcalableACC, DASH, Coarray Fortran, Unified Parallel C (UPC), UPC++, X10, Chapel and so on



Global address space

Private address space

Execution unit

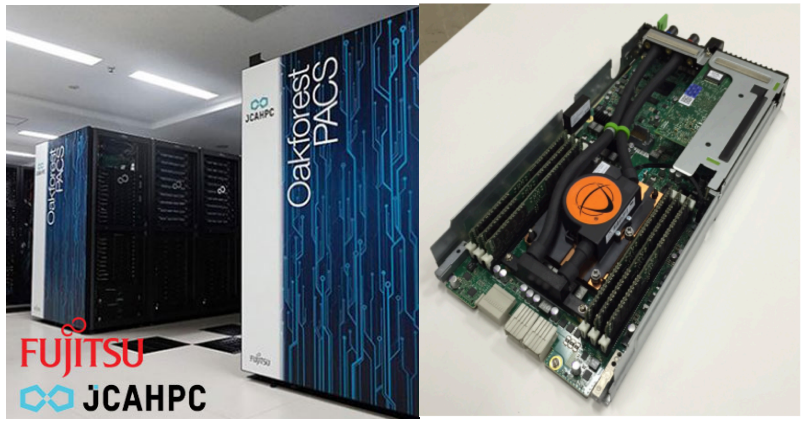
Background (2/2)

- XMP is a directive-based language extension
  - Based on C and Fortran (C++ on the table)
  - Collaborate with OpenMP directives for thread programming
  - Designed by PC cluster consortium
  - http://xcalablemp.org

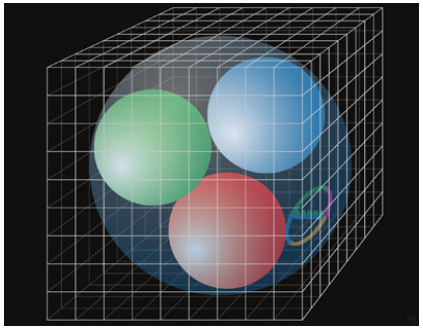
- Omni compiler
  - Reference implementation for XMP
  - Developed by RIKEN AICS and University of Tsukuba
  - Source-to-Source compiler
    - Support : The K computer, Intel Xeon Phi Cluster, Cray machines, ...
  - http://omni-compiler.org

Objective

- Little experience with Omni compiler on Intel Xeon Phi cluster system, which is attracting attention in the HPC field



- Evaluate the performance of Omni compiler on Oakforest-PACS, which is a cluster system based on Knights Landing (9th in the latest Top500 list)
- Make the following key contributions:
  - Evaluation of the Lattice QCD mini-application using a hybrid model of XMP and OpenMP on Oakforest-PACS
  - Effective code translation method for a source-to-source compiler



Agenda from this slide

- Overview of XMP and Omni compiler
- Performance tuning of Omni compiler on a single compute node
- Evaluation of the Lattice QCD mini-application on Oakforest-PACS
- Summary

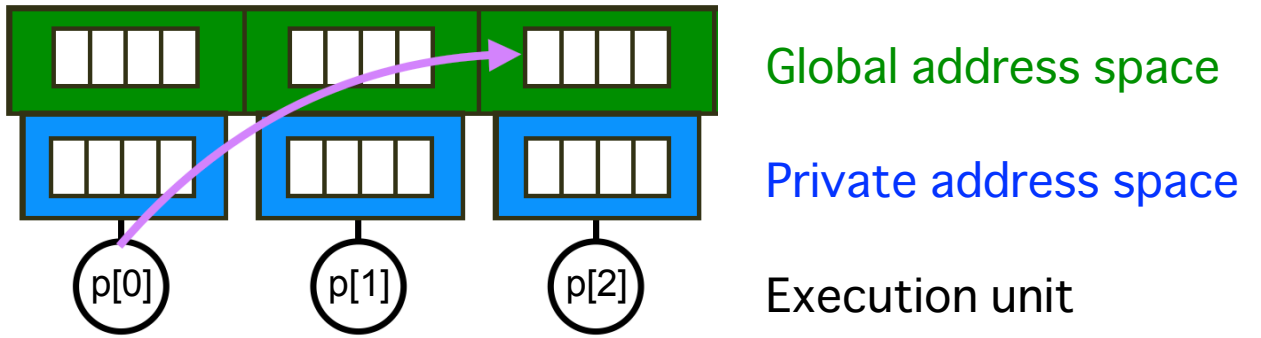
Example of XcalableMP programming

```
int a[MAX];
#pragma xmp nodes p[3]
#pragma xmp template t[MAX]
#pragma xmp distribute t[block] onto p
#pragma xmp align a[i] with t[i]

int main(){
#pragma xmp loop on t[i]
  for(int i = 0; i <MAX; i++)
    a[i] = foo(i);
```

Define execution unit and data distribution

Parallelize loop statement

1. XMP loop directive parallelizes across execution units



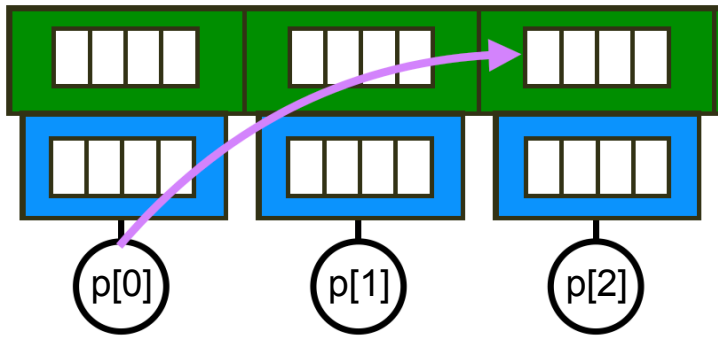
Example of XcalableMP programming

```
int a[MAX];
#pragma xmp nodes p[3]
#pragma xmp template t[MAX]
#pragma xmp distribute t[block] onto p
#pragma xmp align a[i] with t[i]

int main(){
#pragma xmp loop on t[i]
#pragma omp parallel for
  for(int i = 0; i <MAX; i++)
    a[i] = foo(i);
```

Define execution unit and data distribution

Parallelize loop statement

1. XMP loop directive parallelizes across execution units
2. OpenMP parallel for directive parallelizes across threads



Global address space

Private address space

Execution unit

Example of XcalableMP programming

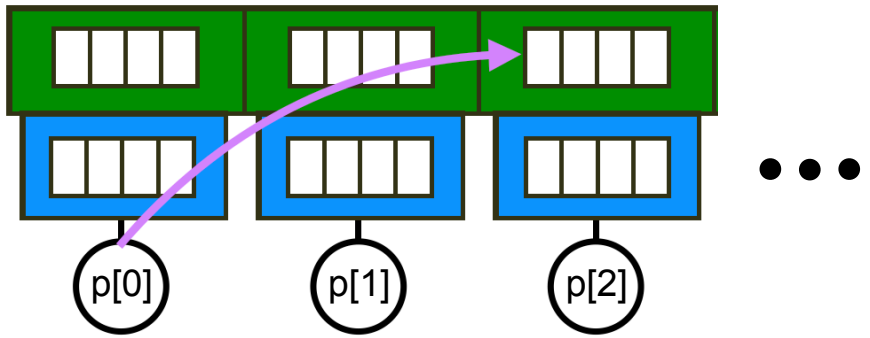
```
int a[MAX];
#pragma xmp nodes p[*]
#pragma xmp template t[MAX]
#pragma xmp distribute t[block] onto p
#pragma xmp align a[i] with t[i]

int main(){
#pragma xmp loop on t[i]
#pragma omp parallel for
  for(int i = 0; i <MAX; i++)
    a[i] = foo(i);
```

Number of execution units is defined dynamically

Define execution unit and data distribution

Parallelize loop statement

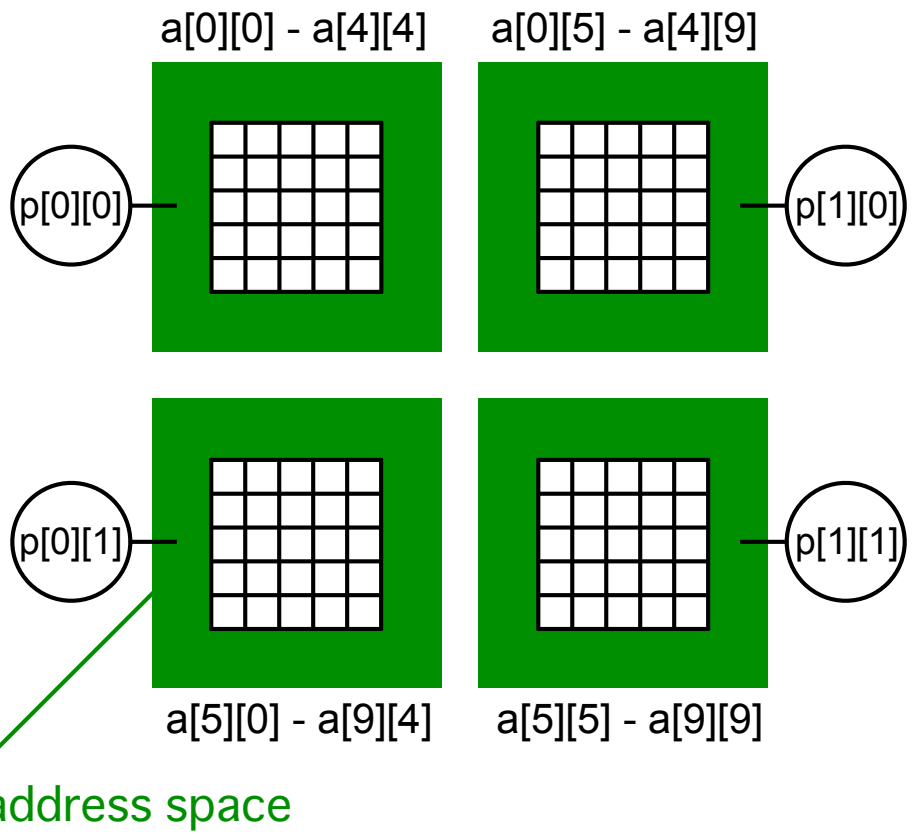
1. XMP loop directive parallelizes across execution units
2. OpenMP parallel for directive parallelizes across threads



Example of XcalableMP programming

- Declaration of multi-dimensional array on global memory address

```
int a[10][10];
#pragma xmp nodes p[2][2]
#pragma xmp template t[10][10]
#pragma xmp distribute t[block][block] onto p
#pragma xmp align a[i][j] with t[i][j]

int main(){
#pragma xmp loop (i,j) on t[i][j]
#pragma omp parallel for collapse(2)
  for(int i = 0; i <10; i++)
    for(int j = 0; j <10; j++)
      a[i][j] = foo(i,j);
```

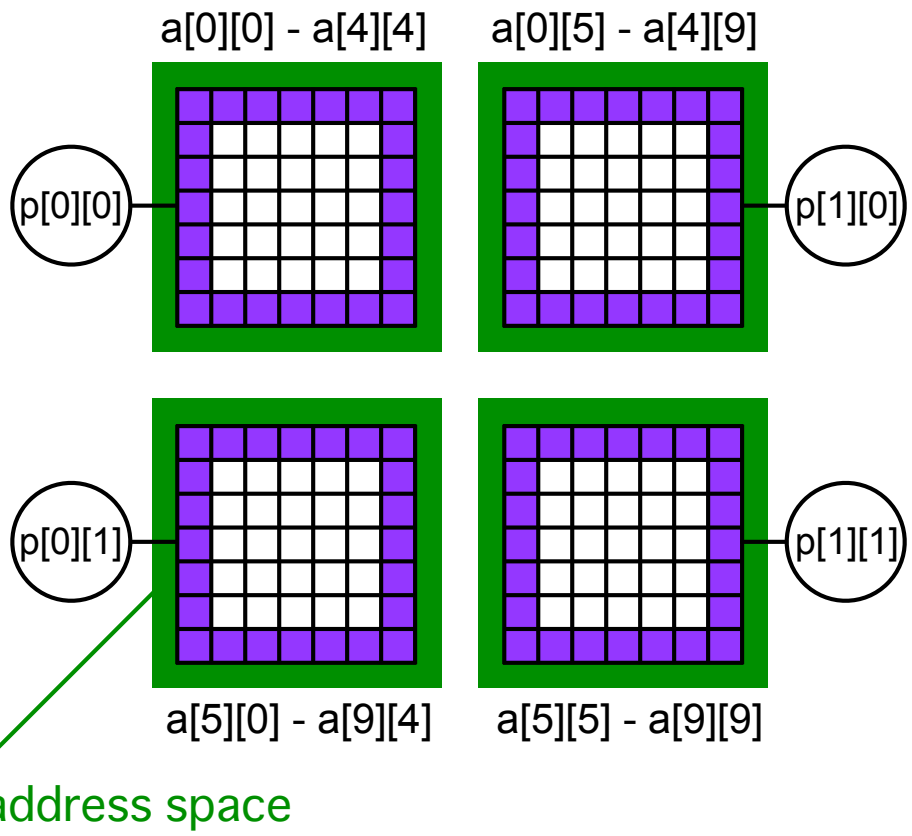


Example of XcalableMP programming

- Shadow/Reflect directives for Stencil application

```
int a[10][10];
#pragma xmp nodes p[2][2]
#pragma xmp template t[10][10]
#pragma xmp distribute t[block][block] onto p
#pragma xmp align a[i][j] with t[i][j]
#pragma xmp shadow a[1][1]

int main(){
  :
#pragma xmp reflect (a) width(/periodic/1) ¥
                        orthogonal
#pragma xmp loop (i,j) on t[i][j]
#pragma omp parallel for
  for(int i = 0; i <10; i++)
    for(int j = 0; j <10; j++)
      ... = a[i-1][j] + a[i+1][j] + a[i][j-1] + a[i][j+1];
```

Shadow directive is to add
halo region in distributed array

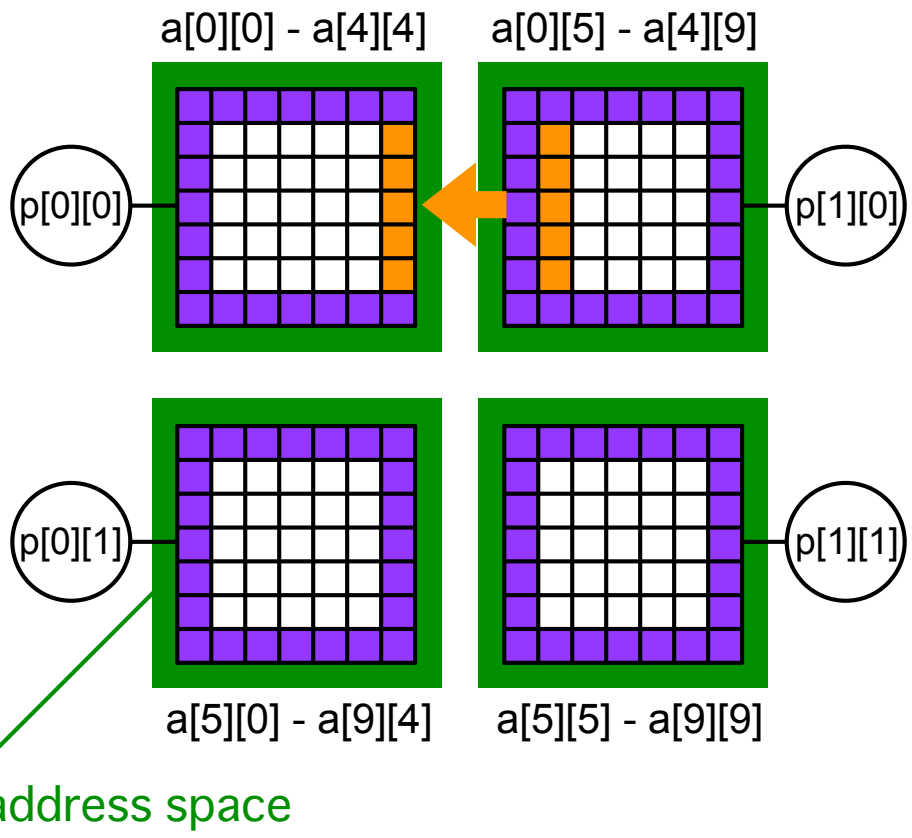


Example of XcalableMP programming

- Shadow/Reflect directives for Stencil application

```
int a[10][10];
#pragma xmp nodes p[2][2]
#pragma xmp template t[10][10]
#pragma xmp distribute t[block][block] onto p
#pragma xmp align a[i][j] with t[i][j]
#pragma xmp shadow a[1][1]

int main(){
  :
#pragma xmp reflect (a) width(/periodic/1) ¥
                        orthogonal
#pragma xmp loop (i,j) on t[i][j]
#pragma omp parallel for
  for(int i = 0; i <10; i++)
    for(int j = 0; j <10; j++)
      ... = a[i-1][j] + a[i+1][j] + a[i][j-1] + a[i][j+1];
```

Reflect directive is to exchange
halo region among neighborhood nodes

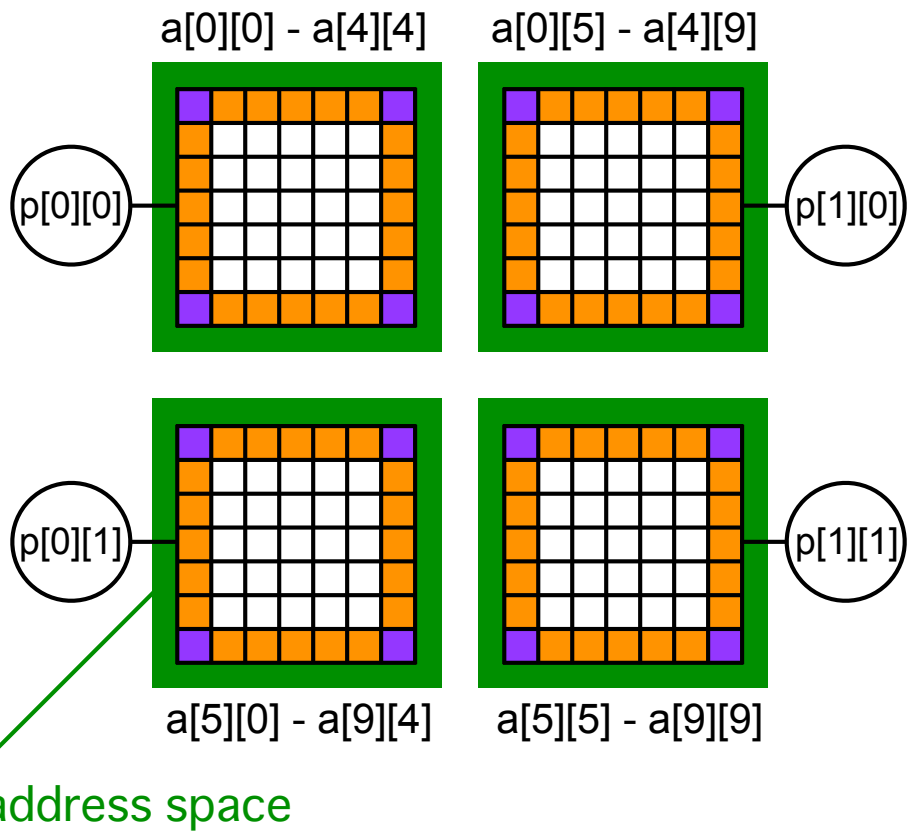


Example of XcalableMP programming

- Shadow/Reflect directives for Stencil application

```
int a[10][10];
#pragma xmp nodes p[2][2]
#pragma xmp template t[10][10]
#pragma xmp distribute t[block][block] onto p
#pragma xmp align a[i][j] with t[i][j]
#pragma xmp shadow a[1][1]

int main(){
  :
#pragma xmp reflect (a) width(/periodic/1) ¥
                        orthogonal
#pragma xmp loop (i,j) on t[i][j]
#pragma omp parallel for
  for(int i = 0; i <10; i++)
    for(int j = 0; j <10; j++)
      ... = a[i-1][j] + a[i+1][j] + a[i][j-1] + a[i][j+1];
```

Reflect directive is to exchange
halo region among neighborhood nodes

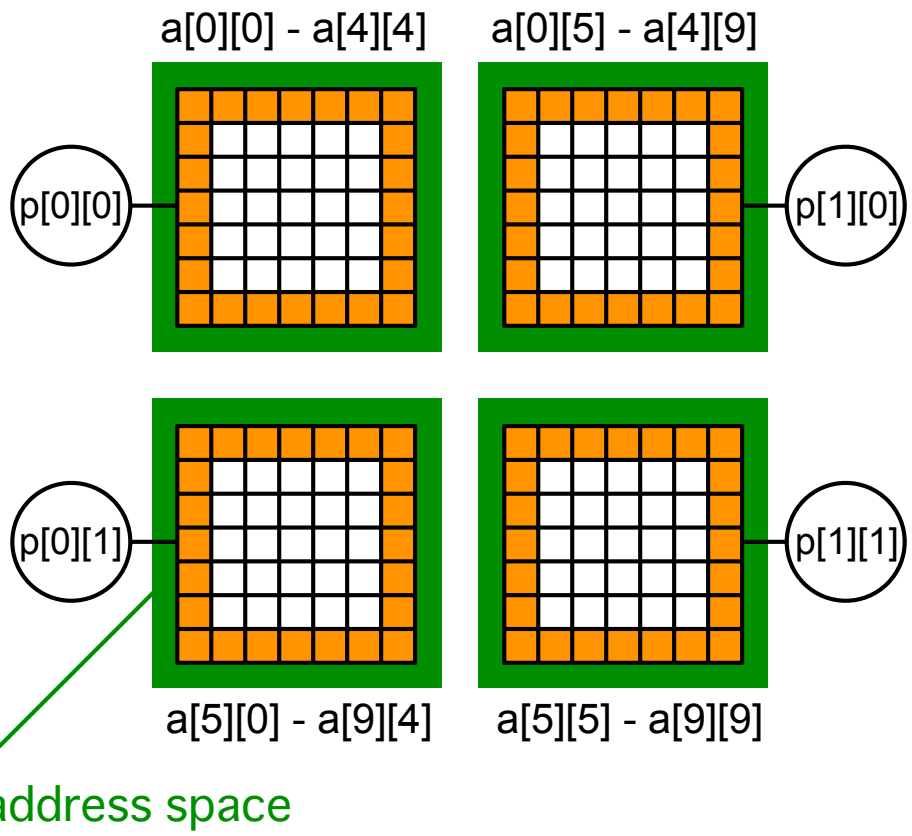


Example of XcalableMP programming

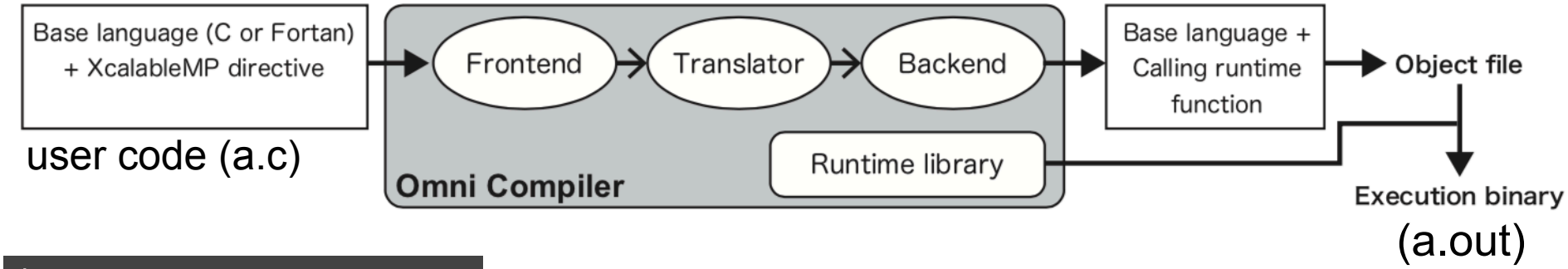
- Shadow/Reflect directives for Stencil application

```
int a[10][10];
#pragma xmp nodes p[2][2]
#pragma xmp template t[10][10]
#pragma xmp distribute t[block][block] onto p
#pragma xmp align a[i][j] with t[i][j]
#pragma xmp shadow a[1][1]

int main(){
  :
#pragma xmp reflect (a) width(/periodic/1) ¥
                        orthogonal
#pragma xmp loop (i,j) on t[i][j]
#pragma omp parallel for
  for(int i = 0; i <10; i++)
    for(int j = 0; j <10; j++)
      ... = a[i-1][j] + a[i+1][j] + a[i][j-1] + a[i][j+1];
```

Reflect directive is to exchange
halo region among neighborhood nodes



Omni compiler




```
$ xmpcc a.c -o a.out
```

- Source-to-Source compiler
  - A user code with XMP directives is translated to a parallel code with runtime calls of Omni compiler's runtime library
  - The Omni compiler's runtime library is implemented in C and MPI
  - The translated parallel code is compiled by a native compiler
    - e.g. GNU, Intel, PGI, Cray, and so on

Examples of code translation

- Define distributed array on global address space

User code

```
double a[10][10];
#pragma xmp align a[i][j] with t[i][j]
```

Translated code

```
int *_XMP_ADDR_a;
          :
_XMP_alloc_array(&_XMP_ADDR_a, ...);
```

Multi-dimensional array is expressed as a pointer and memory is allocated dynamically. One of reasons why memory is allocated dynamically, memory size may be defined dynamically (e.g. using #pragma xmp nodes p[*]).

- Parallelize loop statement on 2x2 execution units

User code

```
#pragma xmp loop (i,j) on t[i][j]
for(int i=0;i<10;i++)
   for(int j=0;j<10;j++)
      a[i][j] = ...;
```

Translated code

```
for(int i=0;i<5;i++)
   for(int j=0;j<5;j++)
      *(_XMP_ADDR_a + i*5 + j) = ..;
```

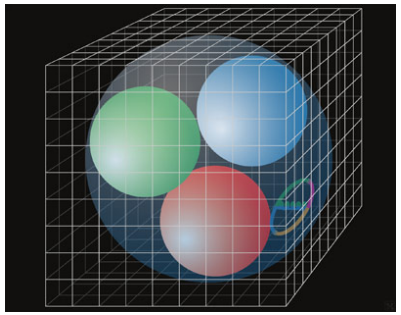
- The initial value and ending condition use constants automatically as possible
- An array operation is translated to a pointer operation

Agenda from this slide

- Overview of XMP and Omni compiler
- Performance tuning of Omni compiler on a single compute node
- Evaluation of the Lattice QCD mini-application on Oakforest-PACS
- Summary

Overview of Lattice QCD

- Lattice QCD is a discrete formulation of QCD (Quantum Chromodynamics)
  - Describe the strong interaction among “quarks” and “gluons”
  - Quark is a species of elementary particles
  - Gluon is a particle that works between quarks



- Lattice QCD is formulated on a four-dimensional lattice (Time and XYZ axes)
- Our Lattice QCD code is based on an existing Lattice QCD mini-application (http://research.kek.jp/people/matufuru/Research/Programs/)
  - By High Energy Accelerator Research Organization, Japan
  - Implemented by extracting the main kernel of the Bridge++, which is a real-world application for lattice gauge theories including QCD (http://bridge.kek.jp/Lattice-code/index_e.html)

Overview of algorithm

- Pseudo-code (CG method is used)

```
S = B                    // COPY
R = B                    // COPY
X = B                    // COPY
sr = norm(S)             // NORM
T = WD(U,X)              // Main Kernel
S = WD(U,T)              // Main Kernel
R = R - S                // AXPY
P = R                    // COPY
rrp = rr = norm(R)       // NORM
do{
  T = WD(U,P)            // Main Kernel
  V = WD(U,T)            // Main Kernel
  pap = dot(V,P)         // DOT
  cr = rr/pap
  X = cr * P + X         // AXPY
  R = -cr * V + R        // AXPY
  rr = norm(R)           // NORM
  bk = rr/rrp
  P = bk * P             // SCAL
  P = P + R              // AXPY
  rrp = rr
}while(rr/sr > 1.E-16)
```

WD() is the Wilson-Dirac operator

$$D_{x,y} = \delta_{x,y} - \kappa \sum_{\mu=1}^{4} \{(1-\gamma_\mu)U_\mu(x)\delta_{x+\hat{\mu},y} + (1+\gamma_\mu)U_\mu^\dagger(x-\hat{\mu})\delta_{x-\hat{\mu},y}\}$$

- Main kernel (most costly)
- Stencil calculation

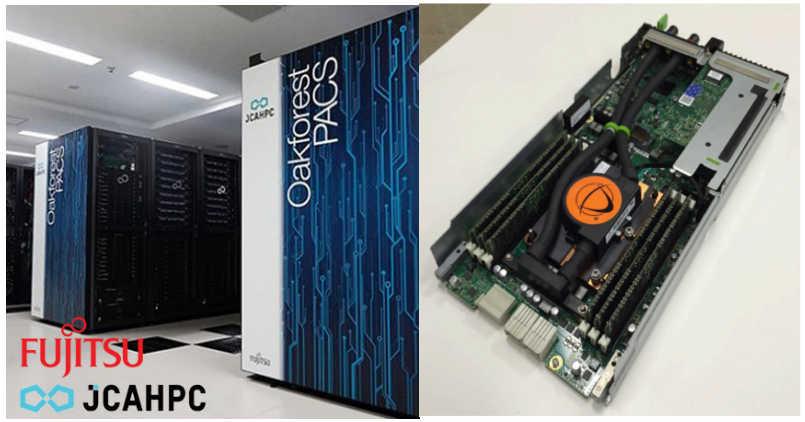
```
#pragma xmp reflect (X) width(..) orthogonal
WD(X, ...);
```

```
void WD(Quark_t X[NT][NZ][NY][NX], ... ){
    :
#pragma xmp loop (t,z) on t[t][z]
#pragma omp parallel for collapse(4)
for(int t=0;t<NT;t++)
  for(int z=0;z<NZ;z++)
    for(int y=0;y<NY;y++)
      for(int x=0;x<NX;x++){
            :
```

Condition of Preliminary Evaluation

- Oakforest-PACS as a KNL cluster system

CPU	Intel Xeon Phi 7250 1.4–1.6GHz 68Cores
Memory	MCDRAM 16GB, DDR4 96GB
Network	Intel Omni-Path Host Fabric Interface 12.5GB/s
Software	intel/2017.4.196, intelmpi/2017.3.196



- COMA as a general PC cluster system

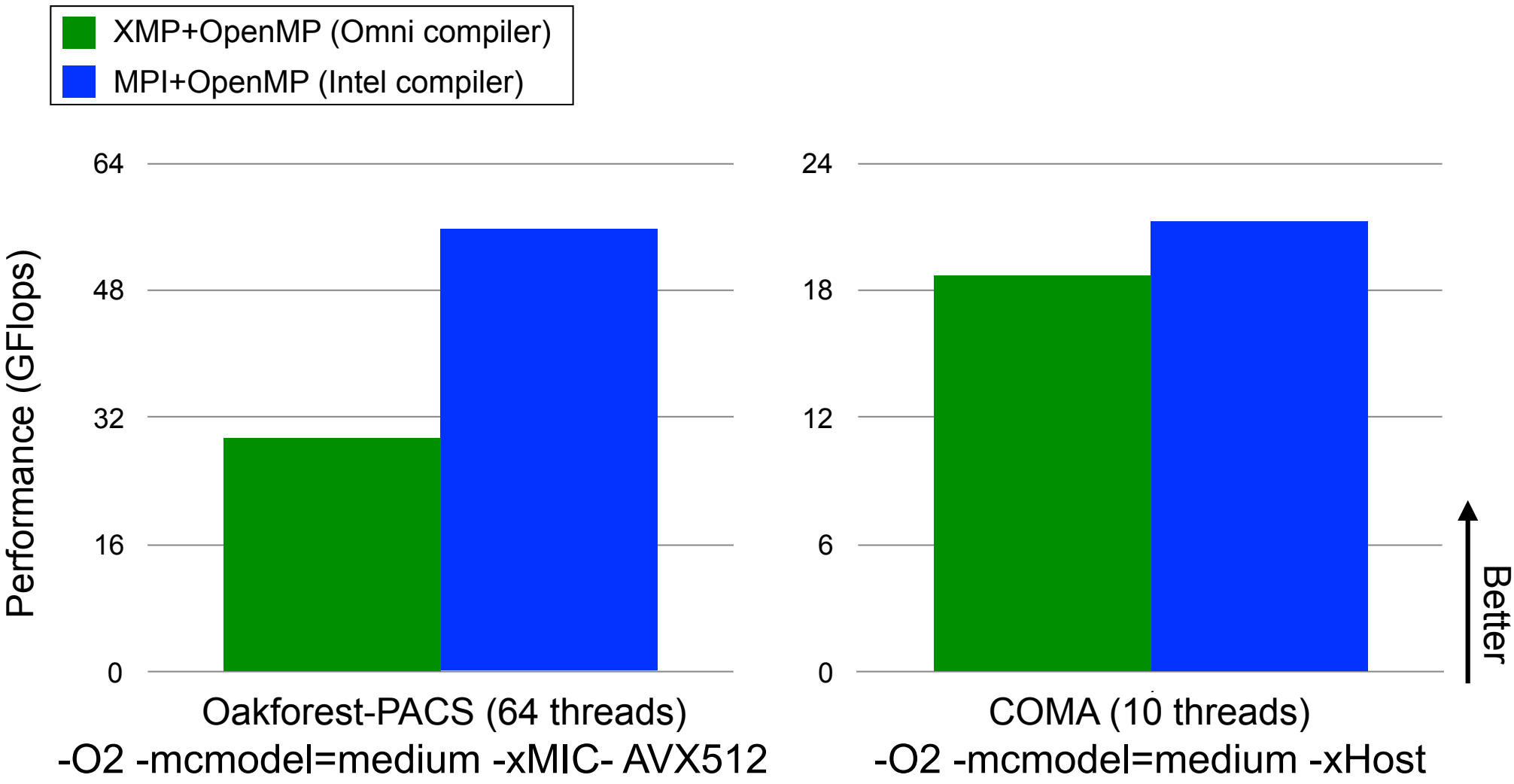
CPU	Intel Xeon-E5 2670v2 2.5–3.3GHz 10Cores, 2Sockets
Memory	DDR3 64GB
Network	InfiniBand FDR 7GB/s
Software	intel/17.0.5, intelmpi/2017.4

Note: Although COMA has KNC, we didn't use it.

For comparison purpose, we also developed Lattice QCD mini-application in MPI+OpenMP.

one process with multi-threads
in a single compute node

Result of Preliminary Evaluation (NT, NZ, NY, NX) = (32,32,32,32)



Although Omni Compiler also uses Intel compiler as a backend compiler, performance results of XMP+OpenMP are worse than those of MPI+OpenMP.

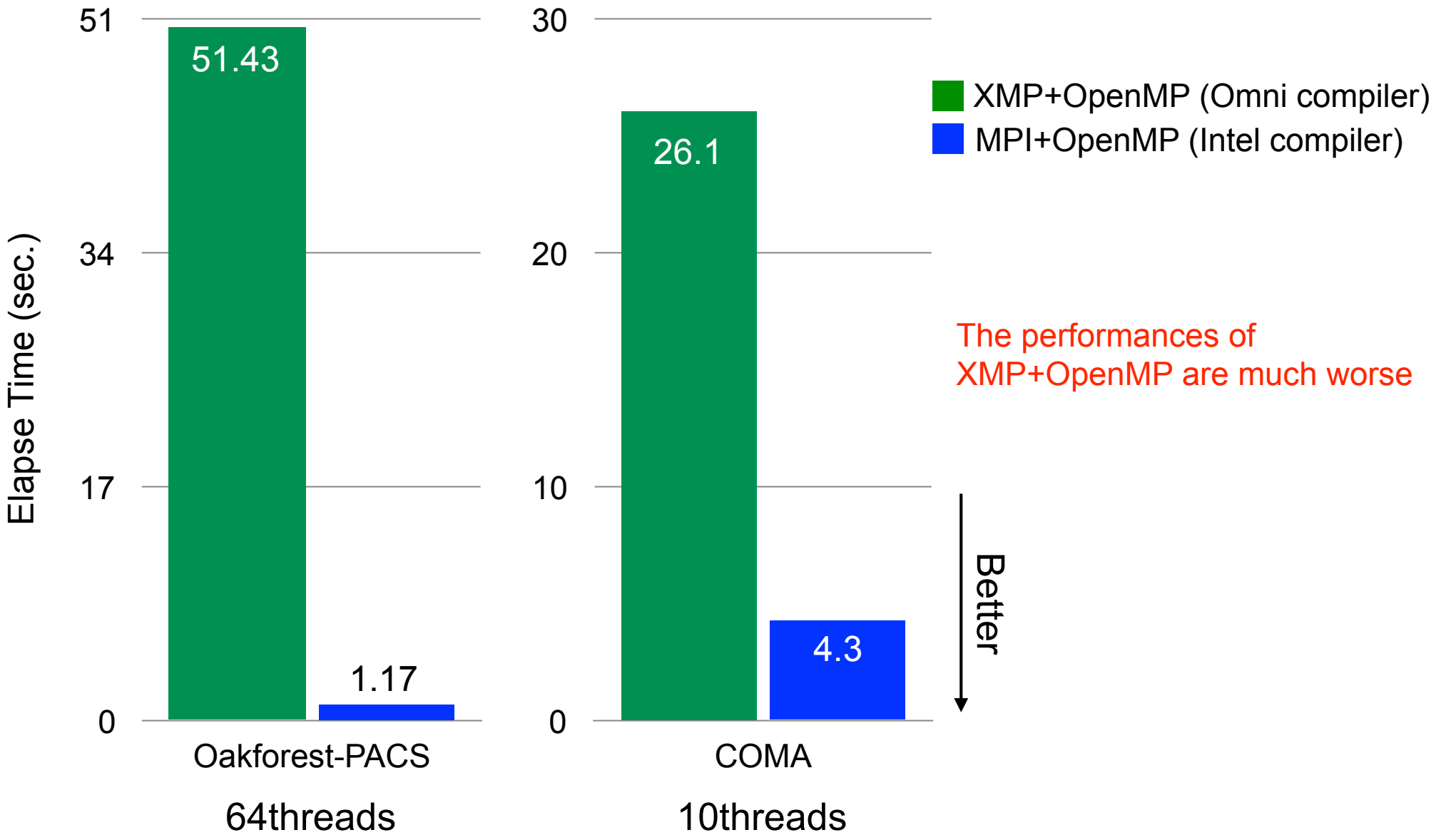
Overview of algorithm

- Pseudo-code (CG method is used)

```
S = B                    // COPY
R = B                    // COPY
X = B                    // COPY
sr = norm(S)             // NORM
T = WD(U,X)              // Main Kernel
S = WD(U,T)              // Main Kernel
R = R - S                // AXPY
P = R                    // COPY
rrp = rr = norm(R)       // NORM
do{
  T = WD(U,P)            // Main Kernel
  V = WD(U,T)            // Main Kernel
  pap = dot(V,P)         // DOT
  cr = rr/pap
  X = cr * P + X         // AXPY
  R = -cr * V + R        // AXPY
  rr = norm(R)           // NORM
  bk = rr/rrp
  P = bk * P             // SCAL
  P = P + R              // AXPY
  rrp = rr
}while(rr/sr > 1.E-16)
```

We profiled all functions.
As a result, we found that the performance results of mathematical functions were worse.
Especially, the performance result of SCAL was much worse.

Result of SCAL (NT, NZ, NY, NX) = (32,32,32,32)



SCAL function (A part of Lattice QCD code)

User code

```
typedef struct Quark {
  double v[4][3][2];
} Quark_t;
Quark_t X[NT][NZ][NY][NX];
#pragma xmp nodes p[1][1]
    :
#pragma xmp shadow (X[1][1][0][0])
```

```
void scal(Quark_t X[NT][NZ][NY][NX],
              const double a){
    :
#pragma xmp loop (t,z) on t[t][z]
#pragma omp parallel for collapse(4)
for(int t=0;t<NT;t++)
  for(int z=0;z<NZ;z++)
    for(int y=0;y<NY;y++)
      for(int x=0;x<NX;x++)
        for(int i=0;i<4;i++)
          for(int j=0;j<3;j++)
            for(int k=0;k<2;k++)
              X[t][z][y][x].v[i][j][k] * = a;
```

Multiply the given vector by the given scalar
(X := a * X)

Tuning Omni compiler

Translated code (old)

```
void scal(Quark_t *_XMP_ADDR_X,
          const double a){
    :
#pragma omp parallel for collapse(4)
for(int t=0;t<NT;t++)
  for(int z=0;z<NZ;z++)
    for(int y=0;y<NY;y++)
      for(int x=0;x<NX;x++)
        for(int i=0;i<4;i++)
          for(int j=0;j<3;j++)
            for(int k=0;k<2;k++)
              (*((*((*(((_XMP_ADDR_X +
              (t+1)*(NZ+2)*(NY)*(NX) +
              (z+1)*(NY)*(NX) + y*(NX) + x)
               ->v) + i)) + j)) + k)) *= a;
```

Translated code (new)

```
void scal(Quark_t *_XMP_ADDR_X,
          const double a){
    :
Quark_t (*X_NEW)[NZ+2][NY][NX] =
  (Quark_t (*)[NZ+2][NY][NX])_XMP_ADDR_X;
#pragma omp parallel for collapse(4)
for(int t=0;t<NT;t++)
  for(int z=0;z<NZ;z++)
    for(int y=0;y<NY;y++)
      for(int x=0;x<NX;x++)
        for(int i=0;i<4;i++)
          for(int j=0;j<3;j++)
            for(int k=0;k<2;k++)
              (*((*((*(((&(X_NEW[t+1][z+1][y][x]))
               ->v) + i)) + j)) + k)) *= a;
```

The reason why the performance is worse is that the size of each dimension has disappeared.

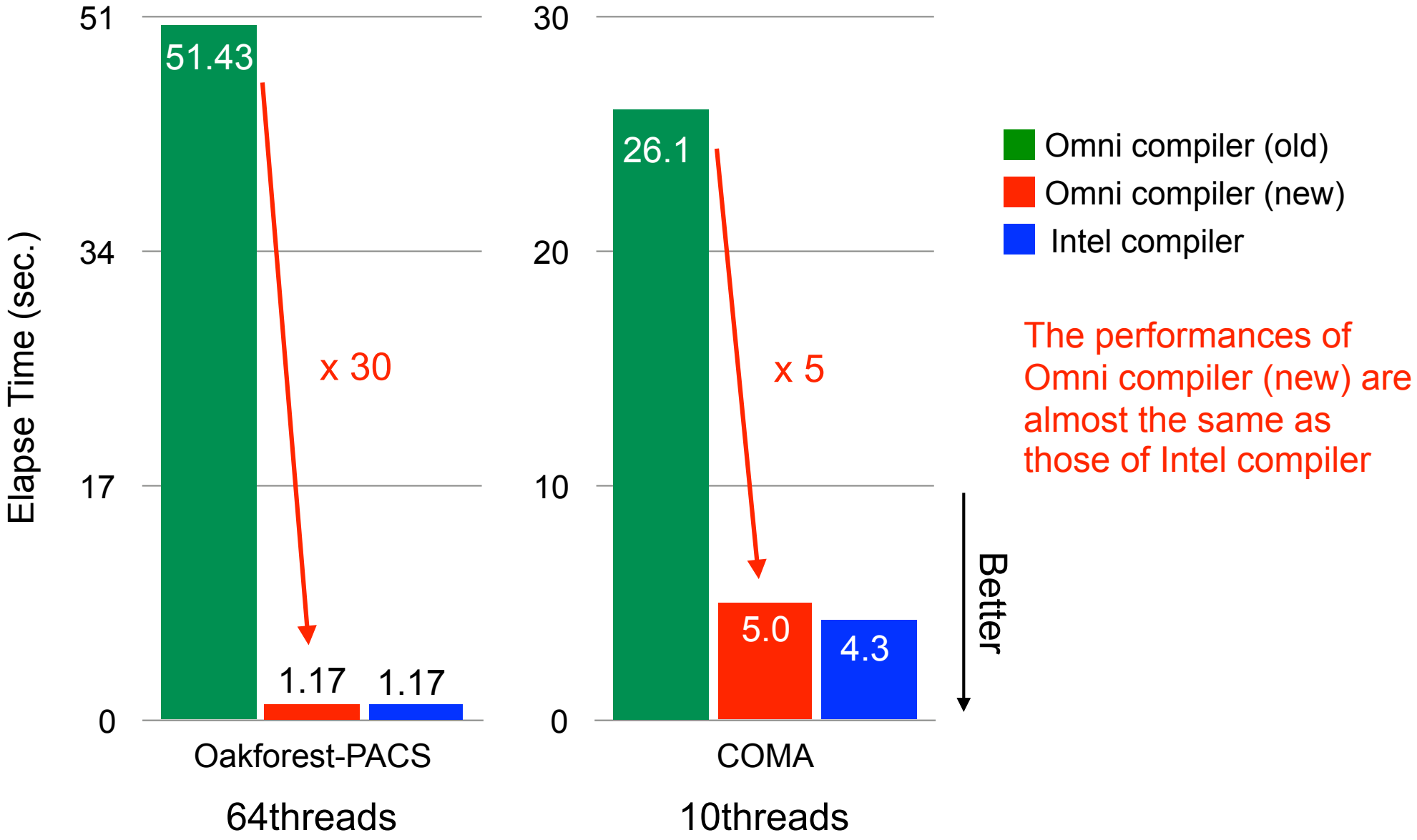
Add a new pointer for a distributed array, which has the size of each dimension.

We add "-qopt-report" option to Intel compiler.

"LOOP WAS NOT VECTORIZED"

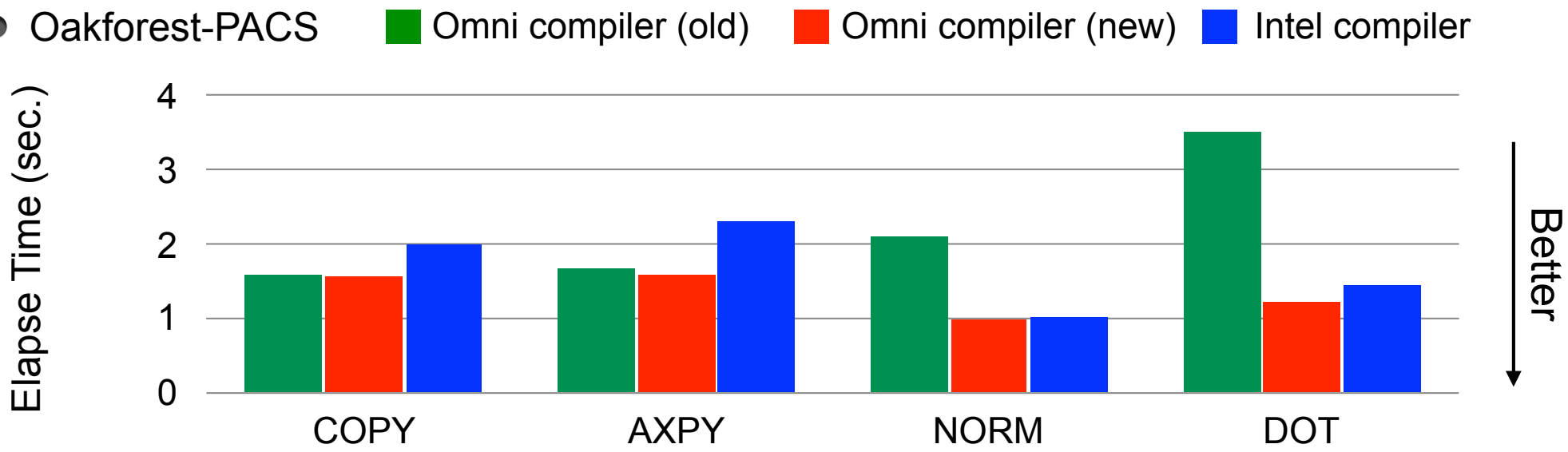
"LOOP WAS VECTORIZED"

Result of SCAL (NT, NZ, NY, NX) = (32,32,32,32)

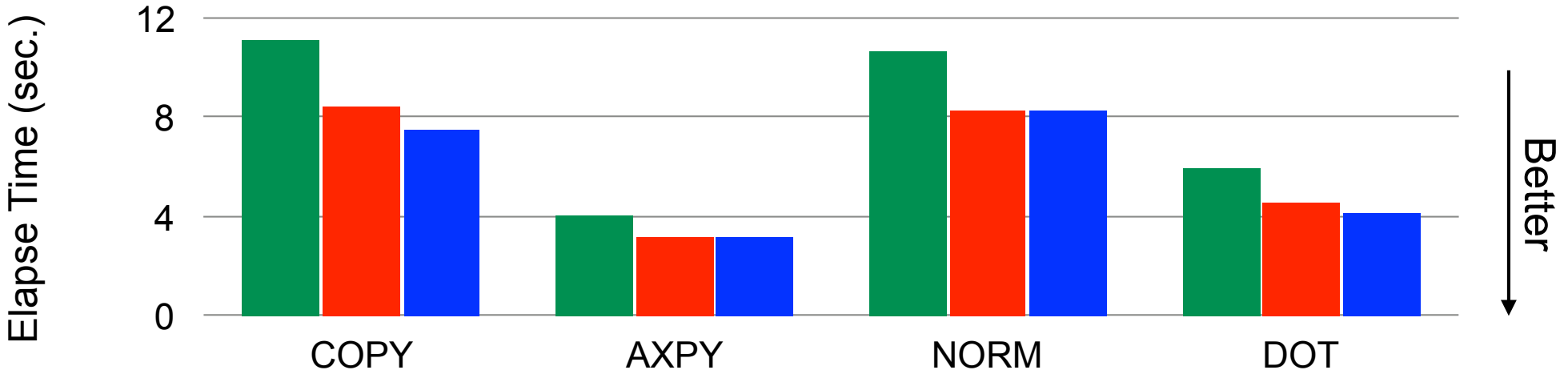


Other mathematical functions (NT, NZ, NY, NX) = (32,32,32,32)

- Oakforest-PACS



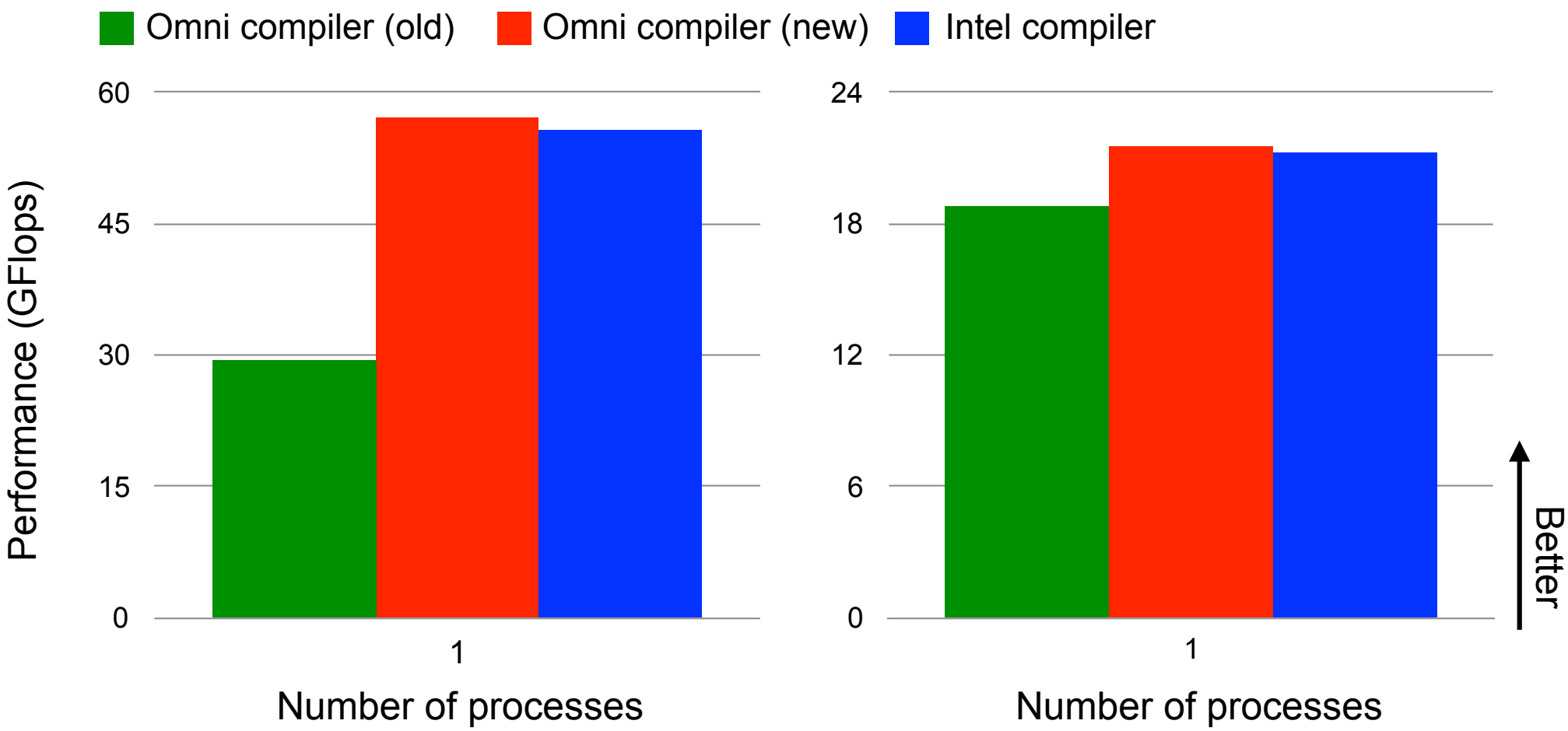
- COMA



The performances of Omni compiler (new) are better than those of Omni compiler (old)

Result of Preliminary Evaluation (NT, NZ, NY, NX) = (32,32,32,32)

- Oakforest-PACS (64 threads)
- COMA (10 threads)



Agenda from this slide

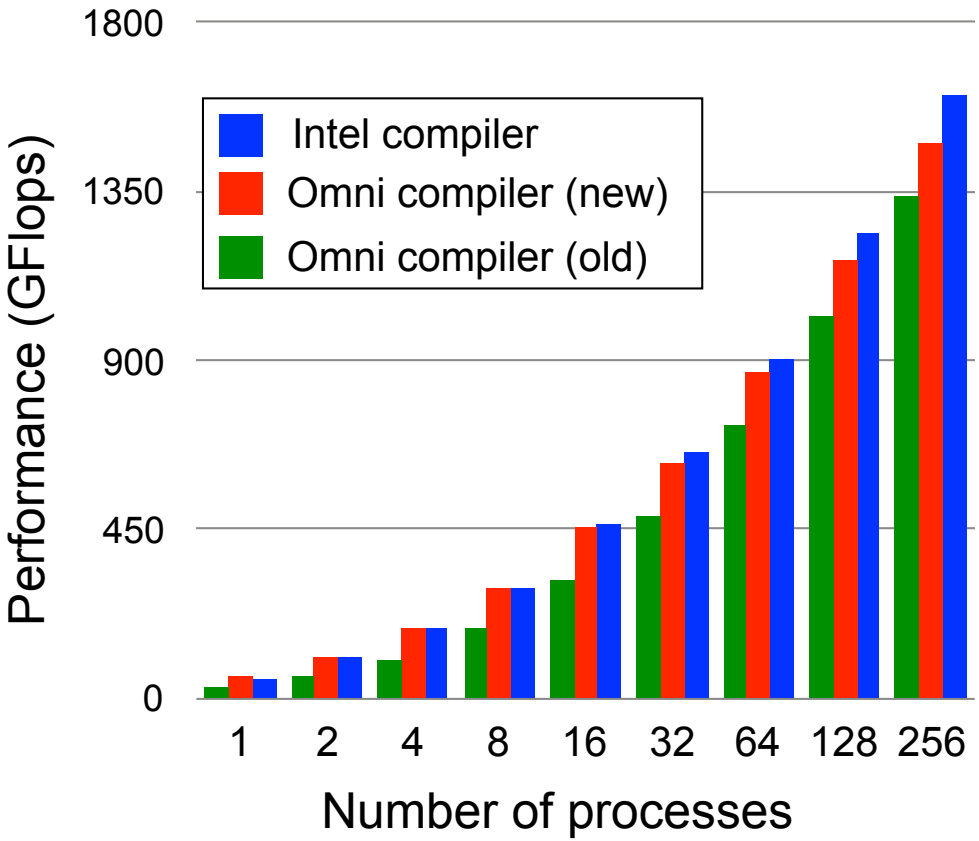
- Overview of XMP and Omni compiler
- Performance tuning of Omni compiler on a single compute node
- Evaluation of the Lattice QCD mini-application on Oakforest-PACS
- Summary

Performance Evaluation on cluster

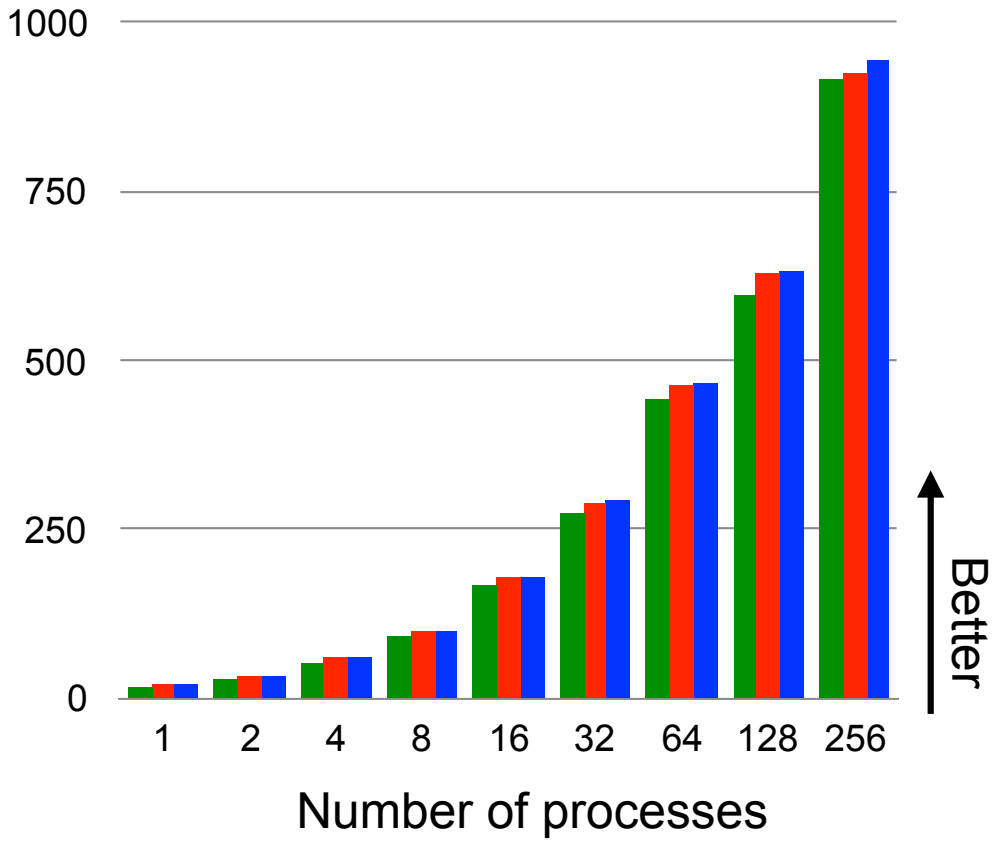
- Oakforest-PACS and COMA
  - One process per compute node on Oakforest-PACS
  - Two processes per compute node on COMA because it has two CPU sockets
- Problem size is (32,32,32,32) as (NT,NZ,NY,NX) with strong scaling

Performance Evaluation

- Oakforest-PACS



- COMA



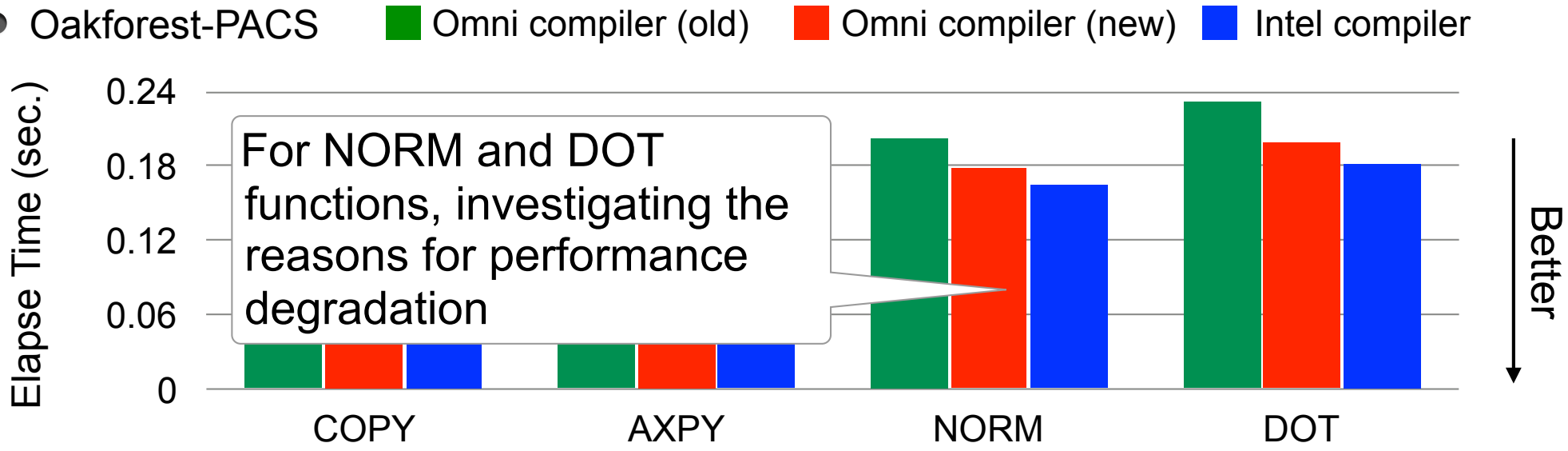
Performances of Omni compiler (new) are always better than those of Omni compiler (old).

Performances of Omni compiler (new) achieve 94 - 105% of those of Intel compiler.

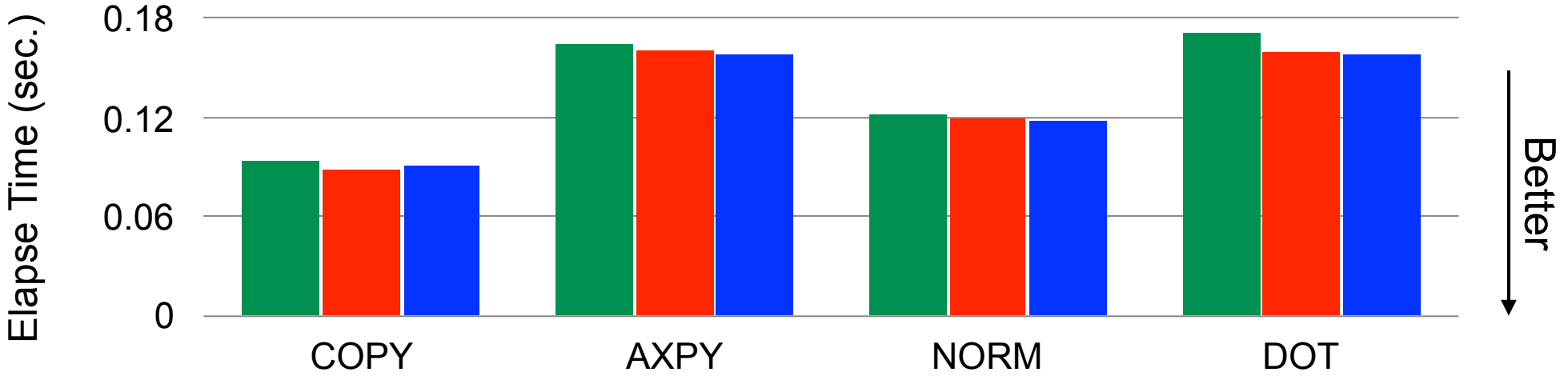
Other mathematical functions (NT, NZ, NY, NX) = (2,2,32,32)

This problem size is a case of using 256 processes.

- Oakforest-PACS

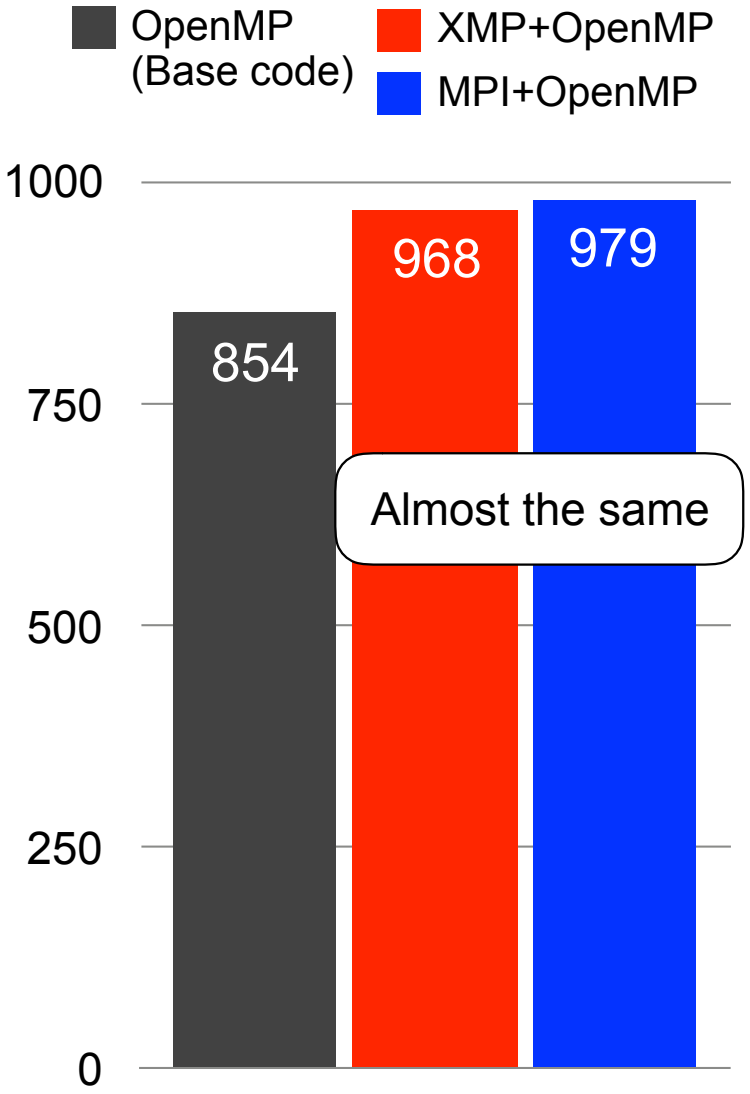


- COMA

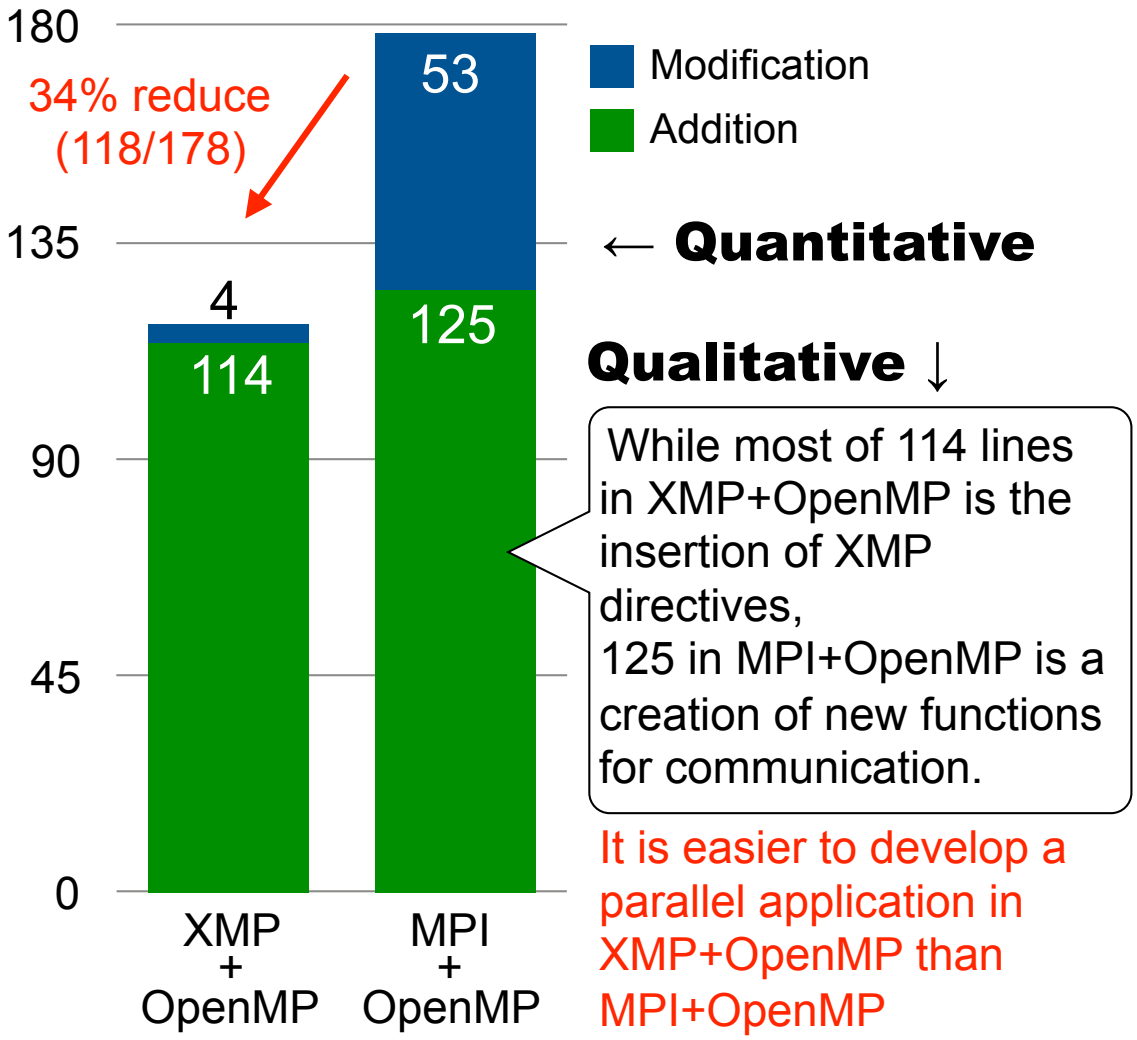


Productivity Evaluation

- Source lines of codes (SLOC)



- Delta SLOC How many lines the code changed from a base code to a parallel code



Agenda from this slide

- Overview of XMP and Omni compiler
- Performance tuning of Omni compiler on a single compute node
- Evaluation of the Lattice QCD mini-application on Oakforest-PACS
- Summary

Conclusion

- We evaluated the performance of the Omni compiler on **Oakforest-PACS**, which is a cluster system based on KNL, and **COMA**, which is a general Linux cluster.
- We tuned performance of Omni compiler
  - The SCAL performance of Omni compiler (new) archives **30 times** better than that of Omni compiler (old) on KNL
- We implemented the Lattice QCD mini-application in XMP+OpenMP
  - The performance in XMP+OpenMP using Omni compiler (new) achieves **94 - 105%** of that in MPI+OpenMP
  - The productivity of XMP+OpenMP is **better** than that of MPI+OpenMP