

Multi-Accelerator Extension in OpenMP based on PGAS Model

Masahiro Nakao, Hitoshi Murai, Mitsuhisa Sato
(RIKEN Center for Computational Science)

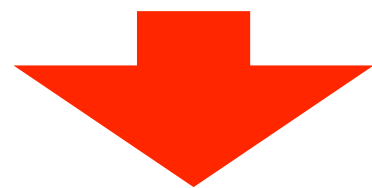
Background

- Accelerators are widely used in various fields
 - e.g. GPU, Xeon Phi, and PEZY-SC2
 - Excellent power performance ratio and memory bandwidth
 - Many systems are ranked in the Top/Green 500 lists
- Single compute node has **multi-accelerator**
 - To improve the power performance ratio further
 - For example, "Summit" in ORNL has six Tesla V100 GPUs per compute node



Motivation

- Accelerator programming languages :
 - CUDA (for only NVIDIA GPU)
 - OpenACC
 - OpenMP (4.0 and later)



Directive-based approach

- High portability since they don't depend on a specific architecture
- Easily develop a parallel code for an accelerator from a sequential code for a host

However, they don't support to deal with multi-accelerator
This is an issue of a programming for the latest architecture

Objective

- We propose to extend **OpenMP syntaxes** to deal with **multi-accelerator**
- Implement some benchmarks using the OpenMP syntaxes, and evaluate them

Agenda

- Background
- Extended OpenMP syntaxes
- Evaluation
- Summary

Comparison of OpenMP and OpenACC

- The syntaxes of OpenMP and OpenACC are very similar

OpenMP

```
double a[N], b[N], c[N], scalar = 1.0;  
#pragma omp target map(to: b,c) map(from: a)  
#pragma omp teams distribute parallel for  
for(int i=0;i<N;i++)  
    a[i] = b[i] + scalar * c[i];
```

OpenACC

```
double a[N], b[N], c[N], scalar = 1.0;  
#pragma acc data copyin(b,c) copyout(a)  
#pragma acc parallel loop  
for(int i=0;i<N;i++)  
    a[i] = b[i] + scalar * c[i];
```

- However, OpenMP has more various functions for a host (CPU thread) than OpenACC
- OpenMP also supports parallelization for shared memory
- To exploit all computing resources of a compute node, it is indispensable to use a host as well as accelerators
- If you use OpenACC, OpenMP is also required for host programming, but when using OpenMP, you need only OpenMP

OpenMP Accelerator Programming

- STREAM Triad for a single accelerator

```
double a[N], b[N], c[N], scalar = 1.0;  
#pragma omp target map(to: b,c) map(from: a)  
#pragma omp teams distribute parallel for  
for(int i=0;i<N;i++)  
    a[i] = b[i] + scalar * c[i];
```

- **target** directive indicates that its range is a kernel for accelerator
 - **map** clause indicates data transfer
 - Transfer the arrays b[] and c[] from host to accelerator before the kernel execution, and the array a[] from accelerator to host after the kernel execution
- **teams distribute parallel for** directive indicates that the following for statement is executed on the accelerator

OpenMP Accelerator Programming

- STREAM Triad for multi-accelerator

```
double a[N], b[N], c[N], scalar = 1.0;
int ndevs = omp_get_num_devices();
assert((N % ndevs) == 0);
int chunk = N / ndevs;
```

```
#pragma omp parallel num_threads(ndevs)
```

```
{
```

```
    int dev_num = omp_get_thread_num();
```

```
    int lb = dev_num * chunk;
```

```
    int ub = lb + chunk;
```

```
#pragma omp target map(to:b[lb:chunk],c[lb:chunk])
```

```
map(from:a[lb:chunk]) device(dev_num)
```

```
#pragma omp teams distribute parallel for
```

```
    for (int i=lb;i<ub;i++)
```

```
        a[i] = b[i] + scalar * c[i];
```

```
}
```

Obtain number of accelerators

Calculate chunk size for each accelerator

Spawns host threads the number of which is the same as the number of accelerators

Transfer only the data necessary for each accelerator

Specify device number

Current Issues

- Necessary to divide data/task to accelerators **manually**
 - Evenly divide data/task and map them appropriately to each accelerator
 - Communication among accelerators may also be required
 - e.g. Halo exchange in a stencil calculation
 - `omp_target_memcpy()` is used to perform the communication
 - Need thread programming to spawn host threads, each thread deals with a single accelerator

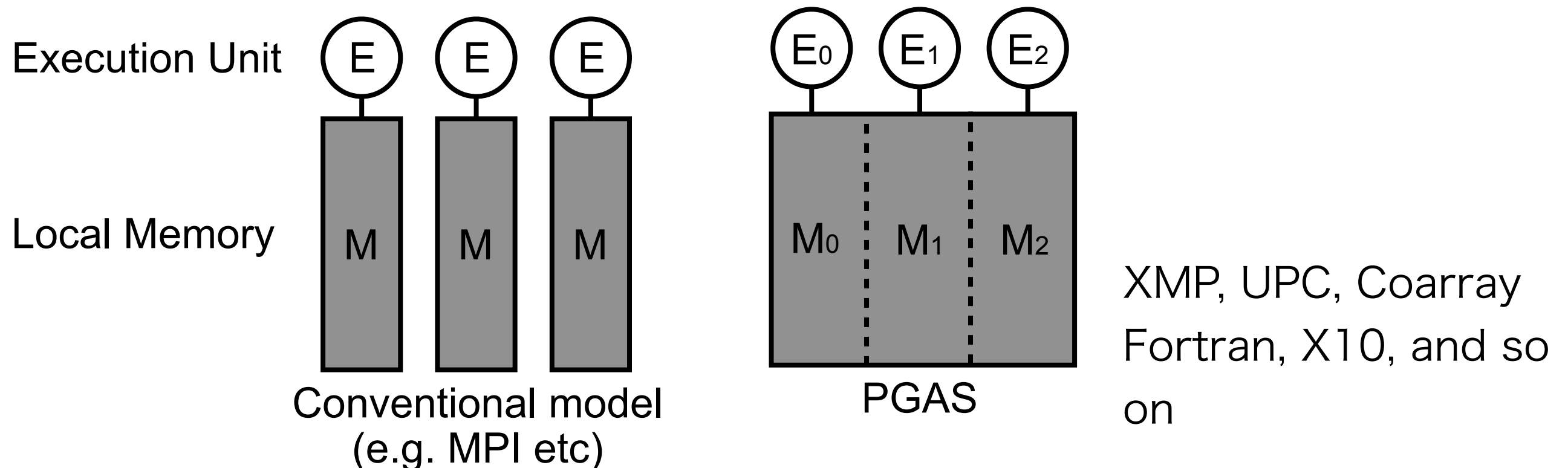


Since the above works are cumbersome, we propose to extend OpenMP syntaxes to make it easy to handle multi-accelerator.

The extended OpenMP directive divides data/task to accelerators **automatically**, and it is easy to generate communication among accelerators.

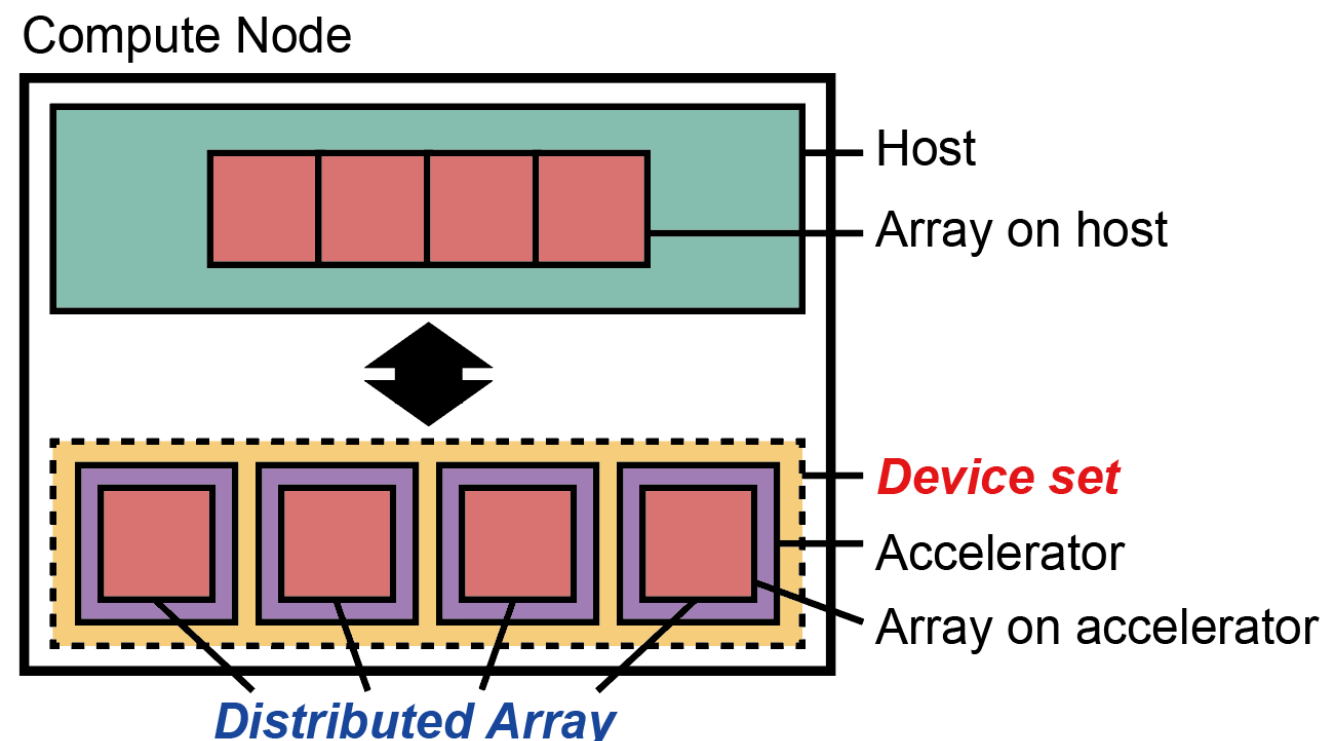
PGAS (Partitioned Global Address Space)

- Parallel programming model for high productivity and performance on distributed memory system
- Provide a global memory space composed of local memories, and each execution unit can freely access the global memory space
- Since an affinity between the local memory and each execution unit is presented, its feature enables users to do programming with data locality awareness



Propose new OpenMP syntaxes

- Features of the new OpenMP syntaxes for multi-accelerator
 - Divide data/task automatically
 - Communication among accelerators easily
- Inherited from XMP (<http://xcalablemp.org>) for C and Fortran
- New terms "**Device set**" and "**Distributed array**"



Device set is a set of multi-accelerator connected to one host.

New OpenMP syntaxes can describe processing like a single computational resource for a **device set**

Distributed array is an array distributed on a **device set**

STREAM Triad using new syntaxes

```
double a[N], b[N], c[N], scalar = 1.0;
#pragma omp target map(to: b,c) map(from: a) device(0:4) layout(block)
#pragma omp teams distribute parallel for device(0:4) layout(block)
for(int i=0;i<N;i++)
    a[i] = b[i] + scalar * c[i];
```

- **device** clause specifies a sub-array as "device(start:length)"
 - A directive with a **device** clause works on its specified devices
- **layout** clause specifies a distribution-manner
 - **block** : block distribution
 - ***** : Not distribute
 - *array-name* : Distribution defined by the specified array
- Support multi-dimensional arrays and nested-loops

```
double d[N][M];
#pragma omp target map(to:d) device(0:2,0:2) layout(block,block)
```

Laplace's equation

```
double u[N][N], v[N][N];
: // Initialize u and v

for(int k=0;k<TIMES;k++){

    for(int j=1;j<N-1;j++)
        for(int i=1;i<N-1;i++)
            v[j][i] = u[j][i];

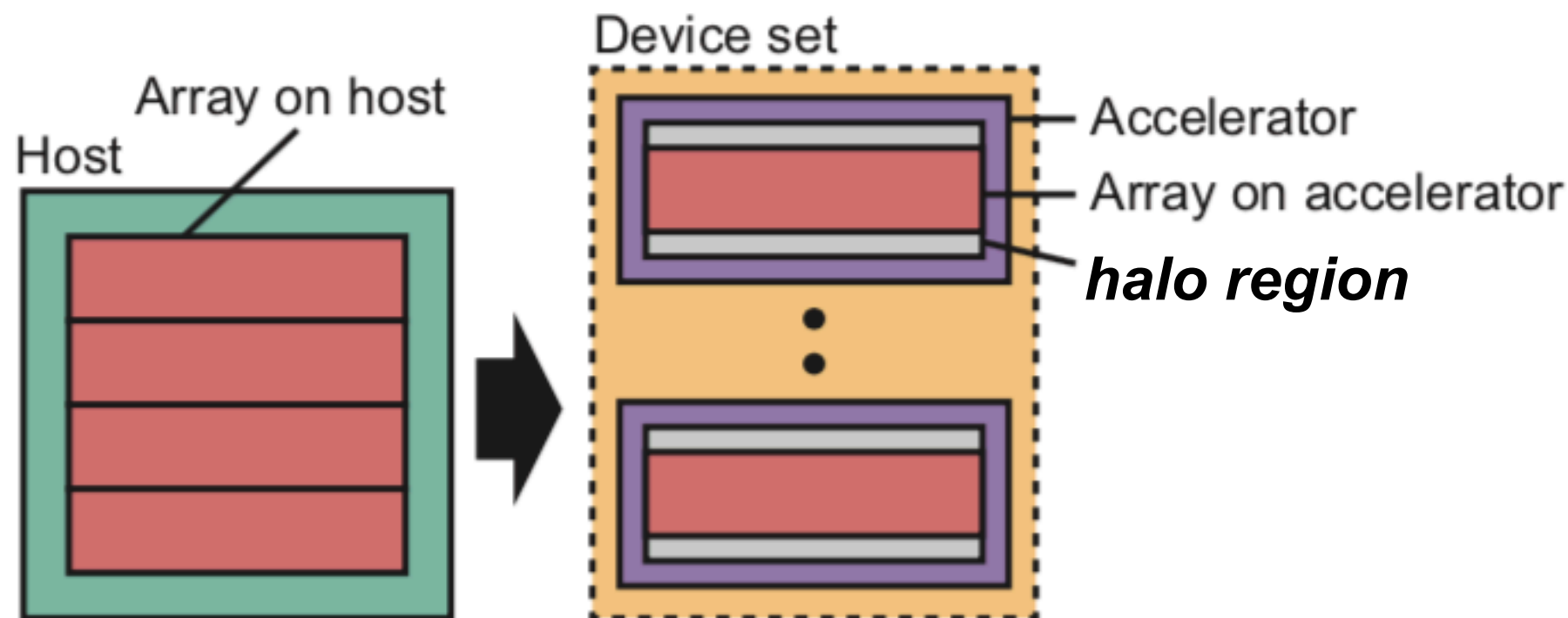
    for(int j=1;j<N-1;j++)
        for(int i=1;i<N-1;i++)
            u[j][i]=(v[j-1][i]+v[j+1][i]+v[j][i-1]+v[j][i+1])/4.0;
}
```

Laplace's equation by new syntaxes

```
double u[N][N], v[N][N];
: // Initialize u and v
#pragma omp target enter data map(to:u) device(0:4) layout(block,*)
#pragma omp target enter data map(to:v) device(0:4) layout(block,*) shadow(1,0)
:
for(int k=0;k<TIMES;k++){
#pragma omp target teams distribute parallel for layout(v)
    for(int j=1;j<N-1;j++)
        for(int i=1;i<N-1;i++)
            v[j][i] = u[j][i];
#pragma omp target reflect(v)
#pragma omp target teams distribute parallel for layout(v)
    for(int j=1;j<N-1;j++)
        for(int i=1;i<N-1;i++)
            u[j][i]=(v[j-1][i]+v[j+1][i]+v[j][i-1]+v[j][i+1])/4.0;
}
#pragma omp target exit data map(from:v) layout(v)
```

Laplace's equation by new syntaxes

```
double u[N][N], v[N][N];  
: // Initialize u and v  
#pragma omp target enter data map(to:u) device(0:4) layout(block,*)  
#pragma omp target enter data map(to:v) device(0:4) layout(block,*) shadow(1,0)
```



- The **device** and **layout** clauses distribute only first dimension of `u[][]` and `v[][]`
- The **shadow** clause adds halo region

Laplace's equation by new syntaxes

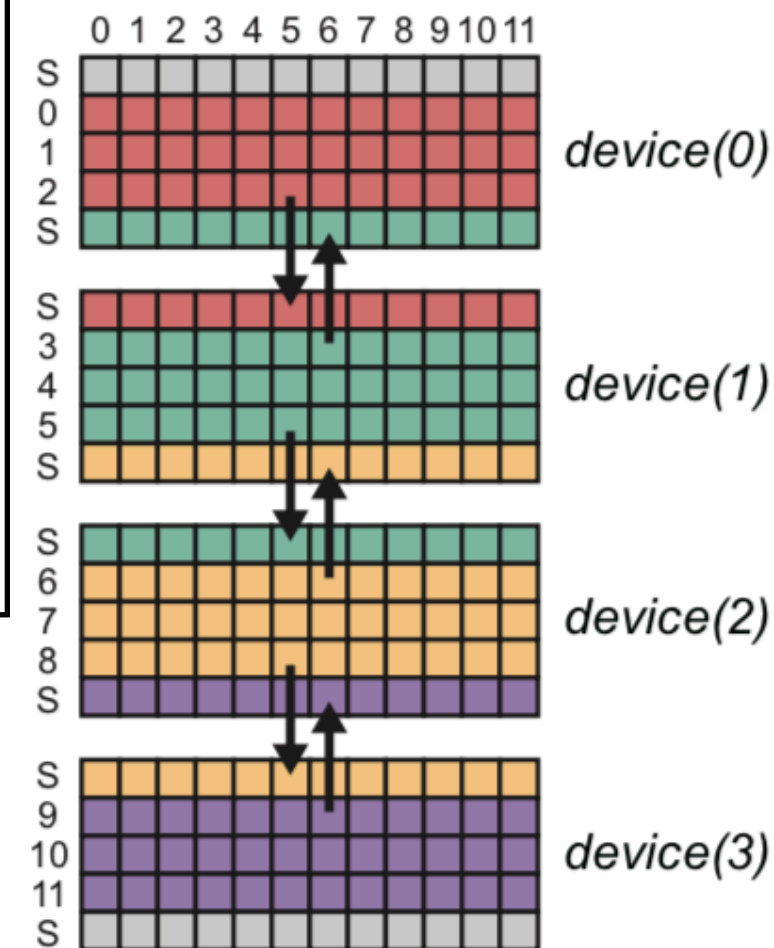
```
for(int k=0;k<TIMES;k++){  
#pragma omp target teams distribute parallel for layout(v)  
    for(int j=1;j<N-1;j++){  
        for(int i=1;i<N-1;i++){  
            v[j][i] = u[j][i];  
#pragma omp target reflect(v)  
#pragma omp target teams distribute parallel for layout(v)  
            for(int j=1;j<N-1;j++){  
                for(int i=1;i<N-1;i++){  
                    u[j][i]=(v[j-1][i]+v[j+1][i]+v[j][i-1]+v[j][i+1])/4.0;  
                }  
#pragma omp target exit data map(from:v) layout(v)  
            }  
        }  
    }
```

target reflect directive updates halo regions
between neighborhood accelerators

Also develop various communication directives

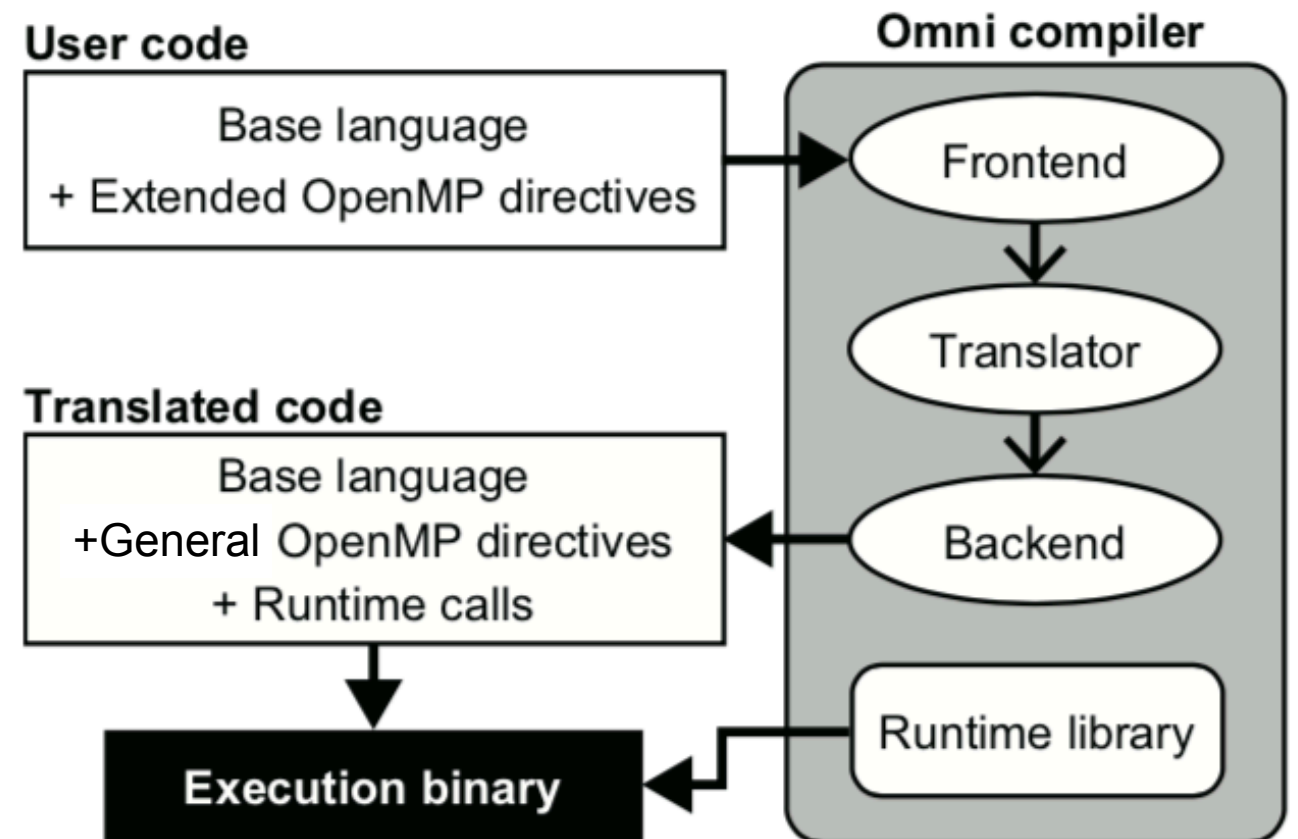
broadcast, reduction, global-array-move (refer to our paper)

Loop-statements are
distributed according
to the distribution
defined by array `v[][]`.




Develop compiler system

- Omni Compiler is a source-to-source compiler infrastructure
 - <http://omni-compiler.org>
- Extended OpenMP syntaxes are translated to general OpenMP syntaxes and a part of code is modified if necessary
- Finally, the backend compiler compiles the translated code to generate the execution binary with linking the runtime library
- (Some features are under construction)



Translated code

```
double a[N], b[N], c[N], scalar = 1.0;
#pragma omp target map(to: b,c) map(from: a) device(0:4) layout(block)
#pragma omp teams distribute parallel for device(0:4) layout(block)
for(int i=0;i<N;i++)
    a[i] = b[i] + scalar * c[i];
```



```
double ** _omni_a = _omni_target_map_start(OMNI_TO, a, sizeof(double), 1,0,4,N, OMNI_BLOCK, ..);
double ** _omni_b = _omni_target_map_start(OMNI_TO, a, sizeof(double), 1,0,4,N, OMNI_BLOCK, ..);
double ** _omni_c; = _omni_target_map_start(OMNI_FROM, a, sizeof(double),1,0,4,N, OMNI_BLOCK, ..);
#pragma omp parallel num_threads(4)
{
    int t = omp_get_thread_num();
    double * _tmp_a = _omni_a[t];
    double * _tmp_b = _omni_b[t];
    double * _tmp_c = _omni_c[t];
    int lb = _omni_get_num_lbound(OMNI_BLOCK,0,N,4,t);
    int ub = _omni_get_num_ubound(OMNI_BLOCK,0,N,4,t);
    #pragma omp target teams distribute parallel for is_device_ptr(_tmp_a, _tmp_b, _tmp_c) device(t)
    for(int j=lb;j<ub;j++)
        _tmp_a[j] = _tmp_b[j] + scalar * _tmp_c[j];
}
```

// Pointers on each devices.

// Lower-/Upper-bounds

Agenda

- Background
- Extended OpenMP Syntaxes
- Evaluation
- Summary

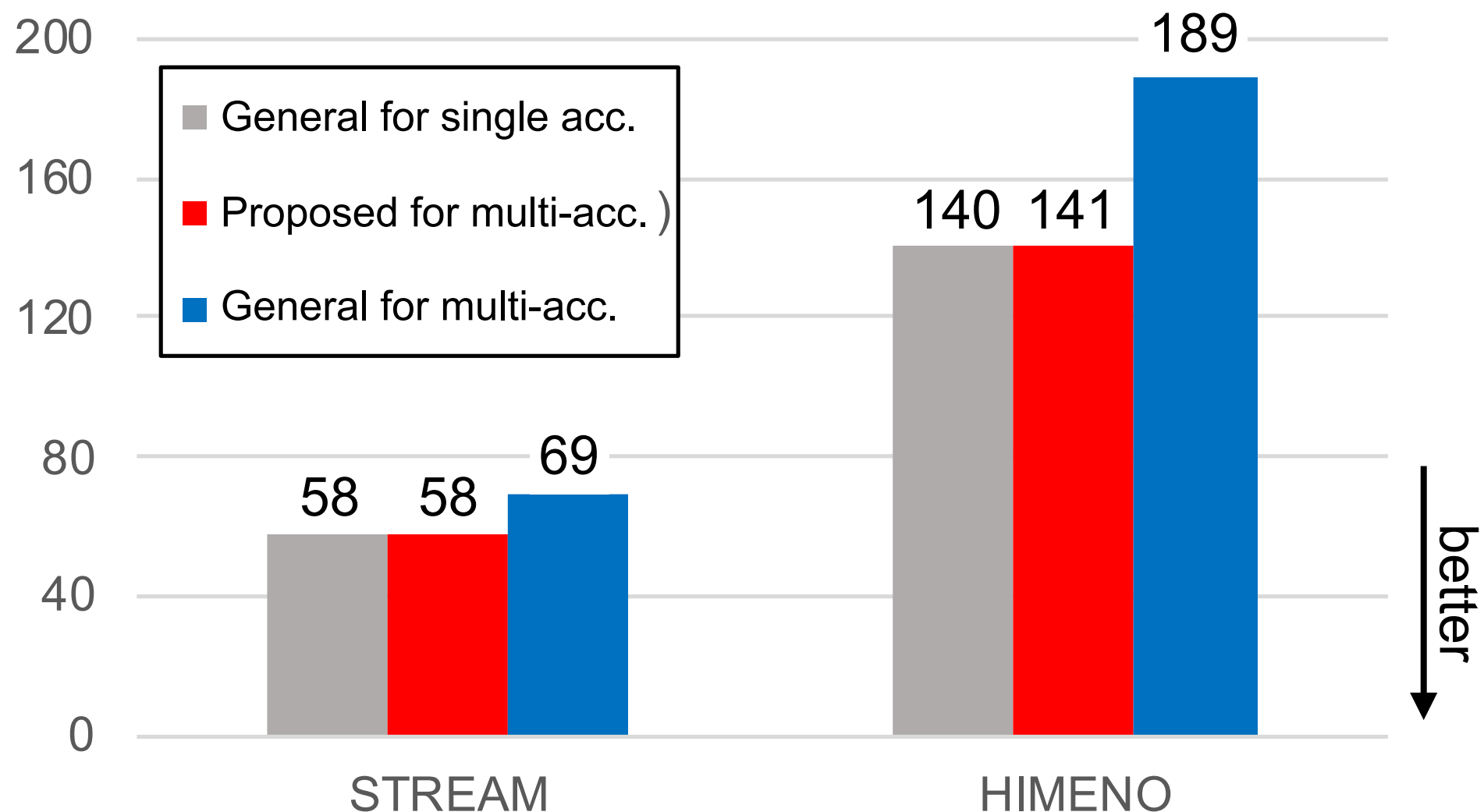
Evaluation of productivity and performance

- STREAM Triad
- HIMENO Benchmark
 - <http://accc.riken.jp/en/supercom/documents/himenobmt/>
 - Evaluates the performance of incompressible fluid analysis code
 - Typical stencil application

CPU	Intel Xeon E5-2698v3 2.3GHz (16 Cores) x 2 Sockets
Memory	DDR3-2133, 128GB, 68 GB/s
GPU	NVIDIA Tesla K80 (GK210 x 2) x 2 GPUs, GDDR5 12GB, 240 GB/s x 2
Software	Clang-ykt (hash:49d8020e03), CUDA 9.1

Productivity (1/2)

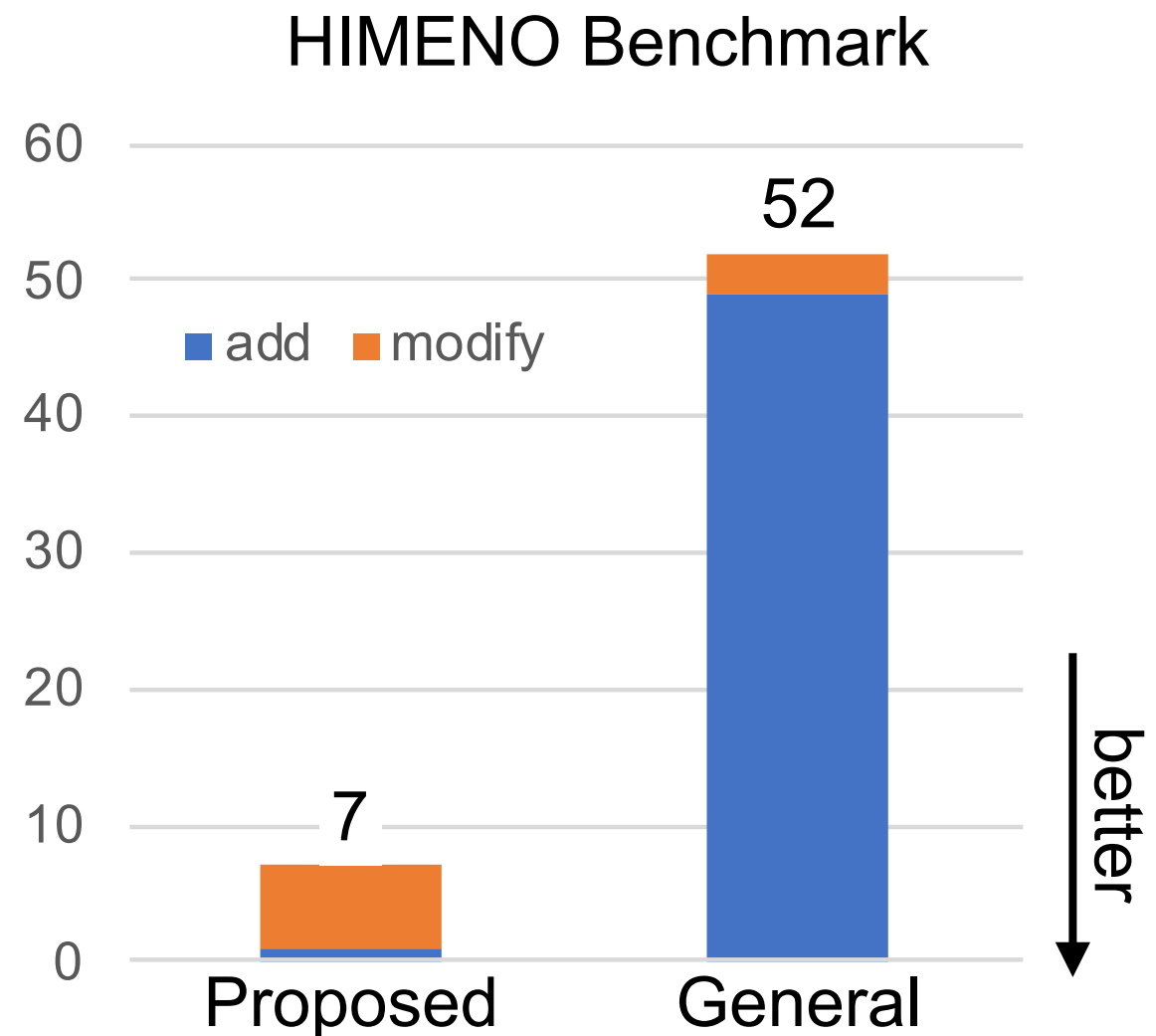
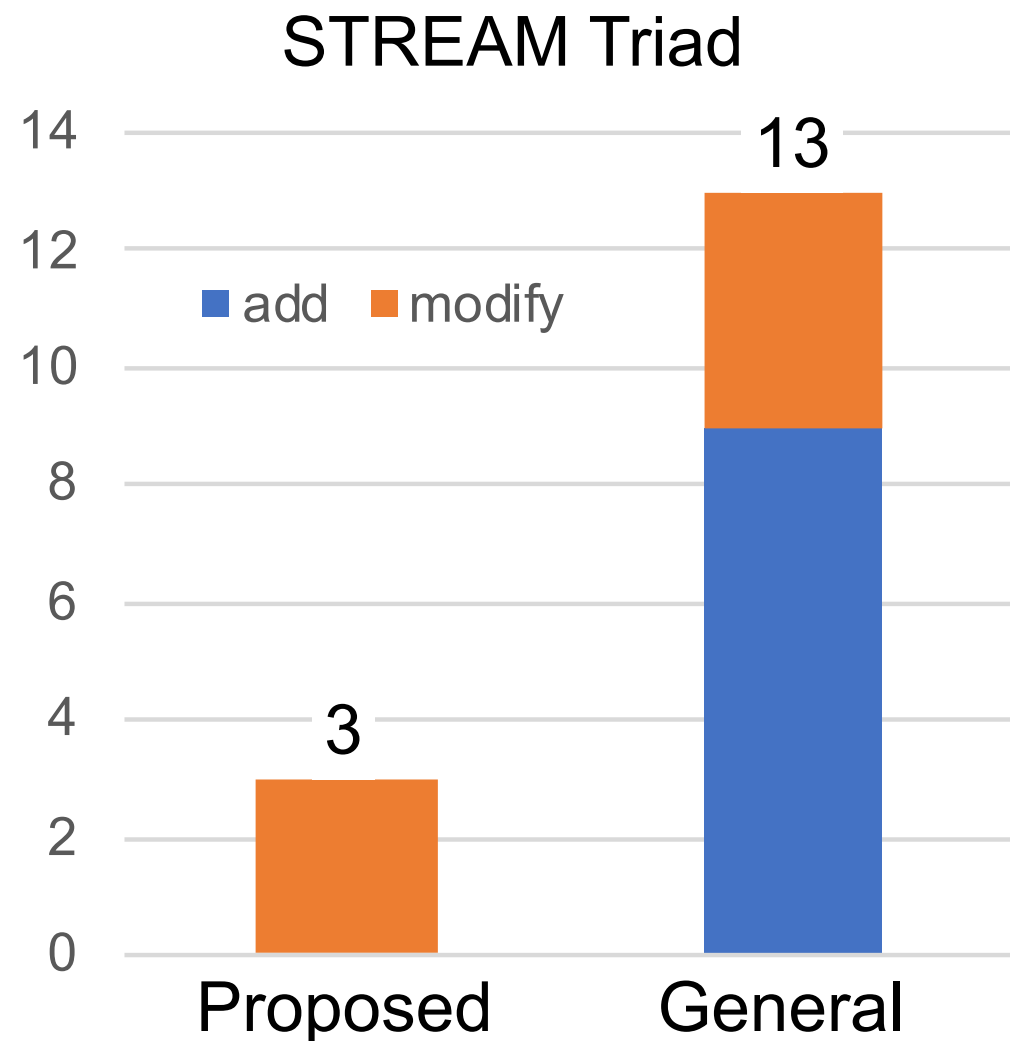
- Source lines of code



The number of lines of the codes using **proposed OpenMP syntaxes** are less than those using **general OpenMP syntaxes**

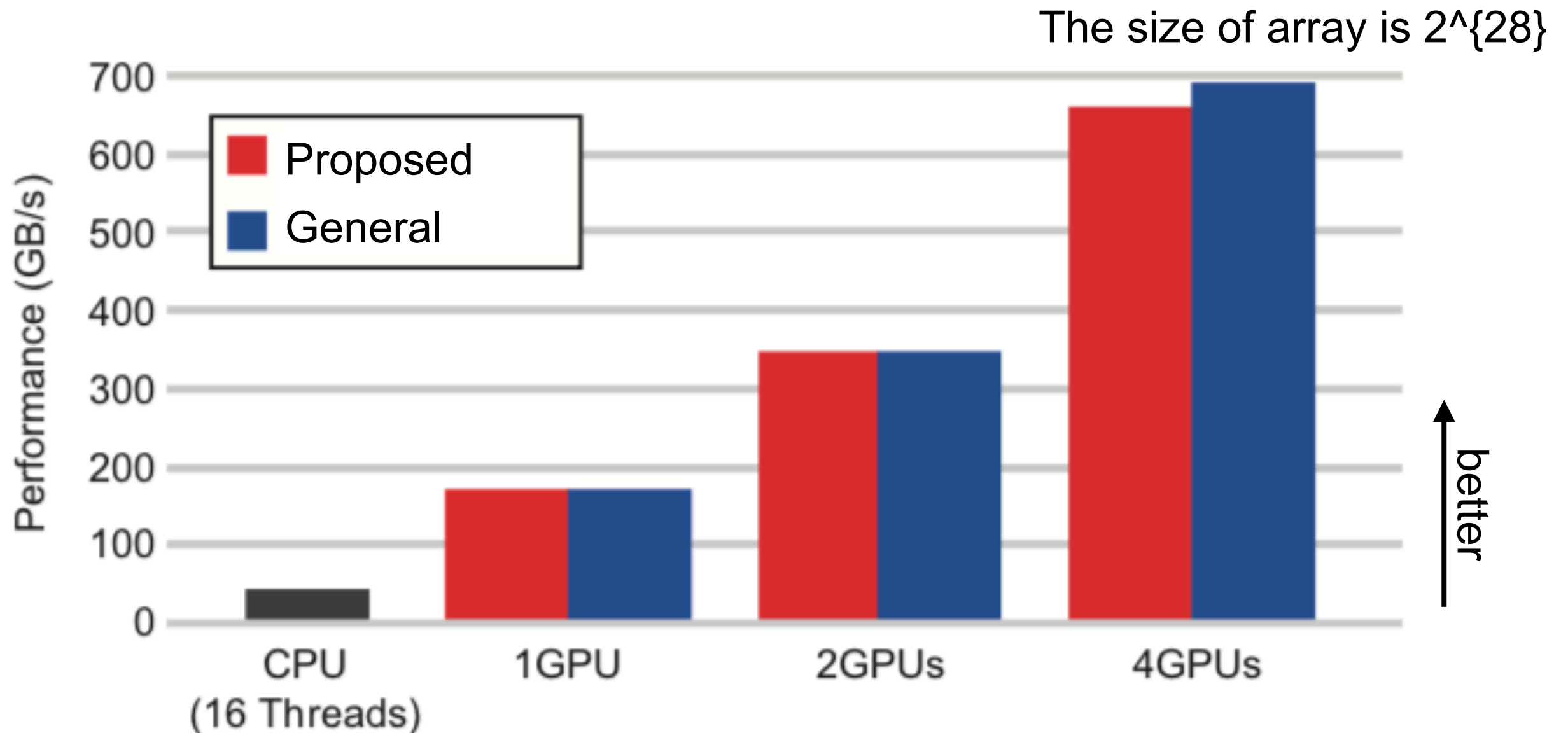
Productivity (2/2)

- How many lines are required to rewrite a code for multi-acc. from that for single acc.



Using proposed OpenMP syntaxes is much less amount of rewriting code than using general OpenMP syntaxes

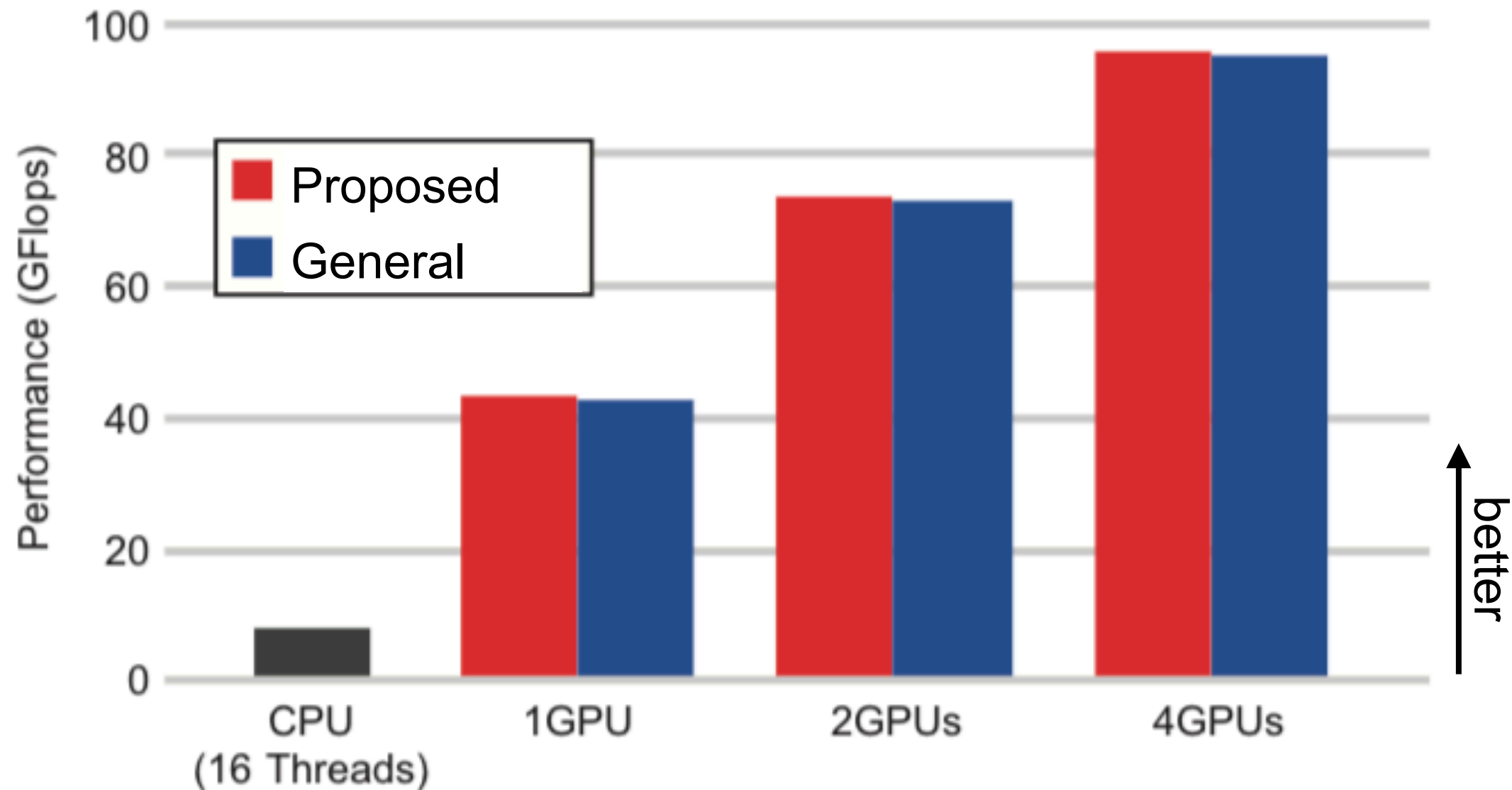
STREAM Triad for performance



The results of “**Proposed**” and “**General**” are almost the same, but only for 4 GPUs, the result of “**Proposed**” is a little worse. The translated code by Omni Compiler is slightly different from manual, so it seems that optimization of the backend compiler (clang-ytk) does not work a little in the benchmark.

HIMENO Benchmark for performance

the problem size is (MIMAX, MJMAX, MKMAX) = (256, 256, 512)



The performance of "**Proposed**" are almost the same as those of "**General**".
(a little better)

Agenda

- Background
- Extended OpenMP Syntaxes
- Evaluation
- Summary

Conclusion

- To further increase productivity for multi-accelerator programming, we propose new OpenMP syntaxes
 - We develop the omni compiler for the proposed OpenMP syntaxes, and implement benchmarks using it
 - The productivity of the proposed OpenMP syntaxes are much better than that of general OpenMP syntaxes
 - The performance of the proposed OpenMP syntax achieved almost the same as that of general OpenMP
- Future work
 - Evaluate more various applications
 - To support accelerated cluster systems, we plan to develop a new language which combines the parallel language XcalableMP and the proposed OpenMP