**What’s XcalableACC?**

- **XcalableACC** is a PGAS language for accelerated clusters, which is a directive-based language extension of C and Fortran (C++ on the table) based on an XcalableMP PGAS language by using OpenACC.
- XcalableACC supports typical parallelization under “global-view model” programming and enables parallelizing the original sequential code by using simple directives.
- XcalableACC also includes coarray features for “local-view model” programming.

**Global-view model**

Array $a[]$ is distributed onto an accelerator memory on each nodes.

```c
int a[12];
#pragma xmp nodes p[3]
#pragma xmp template t[12]
#pragma xmp distribute t[block] onto p
#pragma xmp align a[i] with t[i]
#pragma xmp enter data copyin(a)
```

<table>
<thead>
<tr>
<th>Global index</th>
<th>Distributed Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
<td></td>
</tr>
</tbody>
</table>

Two directives parallelize a for-statement.

```c
#pragma xmp loop on t[i]
#pragma acc parallel loop
for(i = 0; i < 12; i++)
a[i] = i;
```

**Local-view model**

XcalableACC C language defines array section and codimension equivalent to CAF.

```c
x[start:length]:[node_number]
```

Above code means length elements from $x[start]$ to $x[start+length-1]$ located on node_number are referred.

```c
double a[5]*[], b[5]*[]; // Declare
#pragma acc declare create(a, b)

if(xmpc_this_image()==1){
    #pragma acc host_data use_device(a, b)
    b[0:2]:[0] = a[3:2]; // Put
}
```

**Evaluation using Lattice QCD mini-application**

Solve the quantum chromodynamics (QCD) theory of quarks and gluons.

**Productivity**

- Count code changes for developing a parallel code from a serial code (SLOC of a serial code is 842).
- Total code changes of XcalableACC is the smallest of all.

**Performance on HA-PACS/TCA system**

- Each node of HA-PACS/TCA has four GPUs (NVIDIA K20X). We used 256 GPUs on 64 nodes.
- Data size is 32 x 32 x 32 x 32 with strong scaling.
- The performance of XcalableACC is almost the same of those of MPI+OpenACC and MPI+CUDA.