XcalableACC: an Integration of XcalableMP and OpenACC

Akihiro Tabuchi and H. Murai, M. Nakao, T. Odajima, and T. Boku

Abstract XcalableACC (XACC) is an extension of XcalableMP for accelerated clusters. It is defined as a diagonal integration of XcalableMP and OpenACC, which is another directive-based language designed to program heterogeneous CPU/accelerator systems. XACC has features for handling distributed-memory parallelism, inherited from XMP, offloading tasks to accelerators, inherited from OpenACC, and two additional functions: data/work mapping among multiple accelerators and direct communication between accelerators.

Akihiro Tabuchi
Fujitsu Laboratories Ltd., 4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa 211-8588, Japan, e-mail: tabuchi@fujitsu.com??????

Hitoshi Murai
RIKEN Center for Computational Science, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650-0047, Japan, e-mail: h-murai@riken.jp

Masahiro Nakao
RIKEN Center for Computational Science, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650-0047, Japan, e-mail: masahiro.nakao@riken.jp

Tetsuya Odajima
RIKEN Center for Computational Science, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650-0047, Japan, e-mail: tetsuya.odajima@riken.jp

Taisuke Boku
Center for Computationl Sciences, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki 305-8577, Japan, e-mail: taisuke@ccs.tsukuba.ac.jp

Contents

1 Introduction

This document defines the specification of XcalableACC (XACC) which is an extension of XcalableMP version 1.3[1] and OpenACC version 2.5[2]. XcalableACC provides a parallel programming model for accelerated clusters which are distributed memory systems equipped with accelerators.

In this document, terminologies of XcalableMP and OpenACC are indicated by **bold font**. For details, refer to each specification[1, 2].

The works on XACC and the Omni XcalableACC compiler was supported by the Japan Science and Technology Agency, Core Research for Evolutional Science and Technology program entitled "Research and Development on Unified Environment of Accelerated Computing and Interconnection for Post-Petascale Era" in the research area of "Development of System Software Technologies for Post-Peta Scale High Performance Computing."

1.1 Hardware Model

The target of XcalableACC is an accelerated cluster, a hardware model of which is shown in Fig. 1.

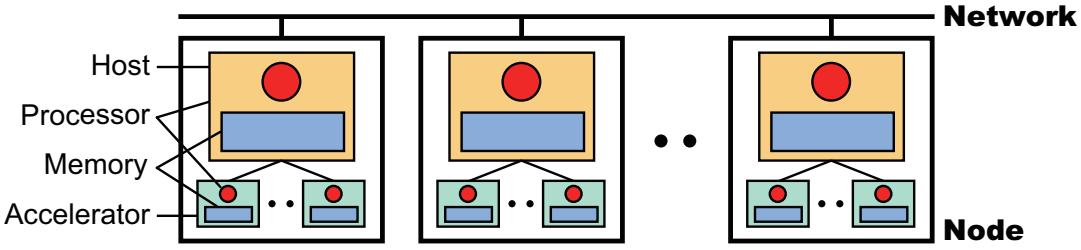

Fig. 1 Hardware Model

An execution unit is called **node** as with XcalableMP. Each **node** consists of a single host and multiple accelerators (such as GPUs and Intel MICs). Each host has a processor, which may have several cores, and own local memory. Each accelerator also has them. Each **node** is connected with each other via network. Each **node** can access its local memories directly and remote memories, that is, the memories of another **node** indirectly. In a host, the accelerator memory may be physically and/or virtually separate from the host memory as with the memory model of OpenACC. Thus, a host may not be able to read or write the accelerator memory directly.

1.2 Programming Model

XcalableACC is a directive-based language extension based on Fortran 90 and ISO C90 (ANSI C90). To develop applications on accelerated clusters with ease, XcalableACC extends XcalableACC and OpenACC independently as follow: (1) XcalableMP extensions are to facilitate cooperation between XcalableMP and OpenACC directives. (2) OpenACC extensions are to deal with multiple accelerators.

1.2.1 XcalableMP Extensions

In a program using the XcalableMP extensions, XcalableMP, OpenACC, and XcalableACC directives are used. Fig. 2 shows a concept of the XcalableMP extensions.

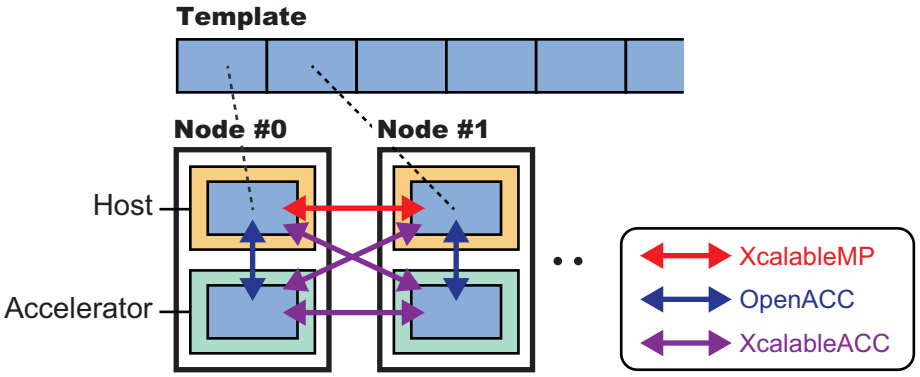

Fig. 2 Concept of XcalableMP Extensions

XcalableMP directives define a **template** and a **node set**. The **template** represents a global index space, which is distributed onto the **node set**. Moreover, XcalableMP directives declare **distributed arrays**, parallelize loop statements and transfer data among host memories according to the distributed **template**. OpenACC directives transfer the **distributed arrays** between host memory and accelerator memory on the same **node** and execute the loop statements parallelized by XcalableMP on accelerators in parallel. XcalableACC directives, which are XcalableMP communication directives with an `acc` clause, transfer data among accelerator memories and between accelerator memory and host memory on different **nodes**. Moreover, **coarray** features also transfer data on different nodes.

Note that the XcalableMP extensions are not a simple combination of XcalableMP and OpenACC. For example, if you represent communication of **distributed array** among accelerators shown in Fig. 2 by the combination of XcalableMP and OpenACC, you need to specify explicitly communication between host and accelerator by OpenACC and that between hosts by XcalableMP. Moreover, you need to calculate

manually indices of the **distributed array** owned by each **node**. By contrast, XcalableACC directives can represent such communication among accelerators directly using global indices.

1.2.2 OpenACC Extensions

The OpenACC extension can represent offloading works and data to multiple-accelerators on a **node**. Fig. 3 shows a concept of the OpenACC extension.

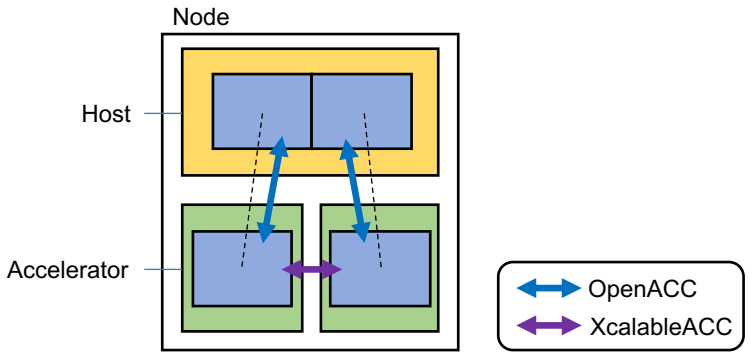

Fig. 3 Concept of OpenACC Extension

OpenACC extension directive defines a **device set**. The **device set** represents a set of devices on a **node**. Futher, OpenACC extension directives declare **distributed arrays** on the **device set** while maintaining the arrays on the host memory, and the directives distribute offloading loop statement and memory copy between host and device memories for the **distributed-arrays**. Moreover, OpenACC extension directives synchronizes devices among the **device set**. XcalableACC directives also transfer data between device memories on the **node**.

1.3 Execution Model

The execution model of XcalableACC is a combination of those of XcalableMP and OpenACC. While the execution model of a host CPU programming is based on that of XcalableMP, that of an accelerator programming is based on that of OpenACC. Unless otherwise specified, each **node** behaves exactly as specified in the XcalableMP specification[1] or the OpenACC specification[2].

An XcalableACC program execution is based on the SPMD model, where each **node** starts execution from the same main routine and keeps executing the same code independently (i.e. asynchronously), which is referred to as the replicated execution

until it encounters an XcalableMP construct or an XcalableMP-extension construct. In particular, the XcalableMP-extension construct may allocate, deallocate, or transfer data on accelerators. An OpenACC construct or an OpenACC-extension construct may define **parallel regions**, such as work-sharing loops, and offloads it to accelerators under control of the host.

When a **node** encounters a loop construct targeted by a combination of XcalableMP `loop` and OpenACC `loop` directives, it executes the loop construct in parallel with other **accelerators**, so that each iteration of the loop construct is independently executed by the **accelerator** where a specified data element resides.

When a **node** encounters a XcalableACC synchronization or a XcalableACC communication directive, synchronization or communication occurs between it and other accelerators. That is, such **global constructs** are performed collectively by the **current executing nodes**. Note that neither synchronizations nor communications occur without these constructs specified.

1.4 Data Model

There are two classes of data in XcalableACC: **global data** and **local data** as with XcalableMP. Data declared in an XcalableACC program are local by default. Both **global data** and **local data** can exist on host memory and accelerator memory. About the data models of host memory and accelerator memory, refer to the OpenACC specification[2].

Global data are ones that are distributed onto the **executing node set** by the `align` directive. Each fragment of a **global data** is allocated in host memory of a **node** in the **executing node set**. OpenACC directives can transfer the fragment from host memory to accelerator memory.

Local data are all of the ones that are not global. They are replicated in the local memory of each of the **executing nodes**.

A **node** can access directly only **local data** and sections of **global data** that are allocated in its local memory. To access data in remote memory, explicit communication must be specified in such ways as the global communication constructs and the **coarray** assignments.

Particularly in XcalableACC Fortran, for common blocks that include any global variables, the ways how the storage sequence of them is defined and how the storage association of them is resolved are implementation-dependent.

1.5 Directive Format

This section describes the syntax and behavior of XcalableMP and OpenACC directives in XcalableACC. In this document, the following notation is used to describe the directives.

`xxx` `type-face` characters are used to indicate literal type characters.
xxx... If the line is followed by "...", then xxx can be repeated.
[xxx] xxx is optional.
`▉` The syntax rule continues.
`[F]` The following lines are effective only in XcalableACC Fortran.
`[C]` The following lines are effective only in XcalableACC C.

In XcalableACC Fortran, XcalableMP and OpenACC directives are specified using special comments that are identified by unique sentinels `!$xmp` and `!$acc` respectively. the directives follow the rules for comment lines of either the Fortran free or fixed source form, depending on the source form of the surrounding program unit[1]. The directives are case-insensitive.

`[F] !$xmp` *directive-name clause*
`[F] !$acc` *directive-name clause*

In XcalableACC, XcalableMP and OpenACC directives are specified using the `#pragma` mechanism provided by the C standards. the directives are case-sensitive.

`[C] #pragma xmp` *directive-name clause*
`[C] #pragma acc` *directive-name clause*

1.6 Organization of This Document

The remainder of this document is structured as follows:

- Chapter 2: XcalableMP Extensions
- Chapter 3: OpenACC Extensions

[1] Consequently, the rules of comment lines that an XcalableMP directive follows is the same as the ones that an OpenMP directive follows.

2 XcalableACC Language

XcalableACC is roughly defined as a diagonal integration of XMP and OpenACC with some additional XACC extensions, where XMP directives are for specifying distributed-memory parallelism, OpenACC for offloading, and the extensions for other XACC-specific features.

The syntax and semantics of XMP and OpenACC directives appearing in XACC codes follow those in XMP and OpenACC, respectively, unless specified below.

2.1 Data Mapping

This chapter defines a behavior of mixing XcalableMP and OpenACC. Note that the existing OpenACC is not extended in the XcalableMP extensions. The XcalableMP extensions can represent (1) parallelization with keeping sequential code image using a combination of XcalableMP and OpenACC, and (2) communication among accelerator memories and between accelerator memory and host memory on different **nodes** using XcalableACC directives or **coarray** features.

When **distributed arrays** appear in OpenACC constructs, global indices in **distributed arrays** are used. The **distributed arrays** may appear in the `update`, `enter data`, `exit data`, `host_data`, `cache`, and `declare` directives, and the `data` clause accompanied by some of `deviceptr`, `present`, `copy`, `copyin`, `copyout`, `create`, and `delete` clauses. Data transfer of **distributed array** by OpenACC is performed on only **nodes** which have elements specified by the global indices.

Example

XcalableACC Fortran

```
integer :: a(N), b(N)
!$xmp template t(N)
!$xmp nodes p(*)
!$xmp distribute t(block) onto p
!$xmp align a(i) with t(i)
!$xmp align b(i) with t(i)
...
!$acc enter data copyin(a(1:K))
!$acc data copy(b)
...
```

XcalableACC C

```
int a[N], b[N];
#pragma xmp template t[N]
#pragma xmp nodes p[*]
#pragma xmp distribute t[block] onto p
#pragma xmp align a[i] with t[i]
#pragma xmp align b[i] with t[i]
...
#pragma acc enter data copyin(a[0:K])
#pragma acc data copy(b)
{ ...
```

Fig. 4 Code example in XcalableMP extensions with `enter_data` directive

In lines 2-6 of Fig. 4, the directives declare the **distributed arrays** *a* and *b*. In line 8, the `enter data` directive transfers the certain range of the **distributed array**

a from host memory to accelerator memory. Note that the range is represented by global indices. In line 9, the `data` directive transfers the whole **distributed array** *b* from host memory to accelerator memory.

2.2 Work Mapping

Description

In order to perform a loop statement on accelerators in **nodes** in parallel, XcalableMP `loop` directive and OpenACC `loop` directive are used. While XcalableMP `loop` directive performs a loop statement in **nodes** in parallel, OpenACC `loop` directive also performs the loop statement parallelized by the XcalableMP `loop` directive on accelerators in parallel. For ease of writing, the order of XcalableMP `loop` directive and OpenACC `loop` directive does not matter.

When `acc` clause appears in XcalableMP loop directive with `reduction` clause, the directive performs a reduction operation for a variable specified in the `reduction` clause on accelerator memory.

Restriction

- In OpenACC **compute region**, only XcalableMP `loop` directive without `reduction` clause can be inserted.
- In OpenACC **compute region**, targeted loop condition (lower bound, upper bound, and step of the loop) must remain unchanged.
- `acc` clause in XcalableMP loop directive can appear only when `reduction` clause appears there.

Example 1

In lines 2-6 of Fig. 5, the directives declare **distributed arrays** *a* and *b*. In line 8, the `parallel` directive with the `data` clause transfers the **distributed arrays** *a* and *b* from host memory to accelerator memory. Moreover, in lines 8-9, the `parallel` directive and XcalableMP `loop` directive perform the next loop statement on accelerators in **nodes** in parallel.

Example 2

In lines 2-5 of Fig. 6, the directives declare **distributed array** *a*. In line 7, the `parallel` directive with the `data` clause transfers the **distributed array** *a*

XcalableACC Fortran

```
integer :: a(N), b(N)
!$xmp template t(N)
!$xmp nodes p(*)
!$xmp distribute t(block) onto p
!$xmp align a(i) with t(i)
!$xmp align b(i) with t(i)
...
!$acc parallel loop copy(a, b)
!$xmp loop on t(i)
do i=0, N
  b(i) = a(i)
end do
!$acc end parallel
```

XcalableACC C

```
int a[N], b[N];
#pragma xmp template t[N]
#pragma xmp nodes p[*]
#pragma xmp distribute t[block] onto p
#pragma xmp align a[i] with t[i]
#pragma xmp align b[i] with t[i]
...
#pragma acc parallel loop copy(a, b)
#pragma xmp loop on t[i]
for(int i=0;i<N;i++){
  b[i] = a[i];
}
```

Fig. 5 Code example in XcalableMP extensions with OpenACC loop construct

XcalableACC Fortran

```
integer :: a(N), sum = 10
!$xmp template t(N)
!$xmp nodes p(*)
!$xmp distribute t(block) onto p
!$xmp align a(i) with t(i)
...
!$acc parallel loop copy(a, sum) reduction(+:sum)
!$xmp loop on t(i) reduction(+:sum) acc
do i=0, N
  sum = sum + a(i)
end do
!$acc end parallel loop
```

XcalableACC C

```
int a[N], sum = 10;
#pragma xmp template t[N]
#pragma xmp nodes p[*]
#pragma xmp distribute t[block] onto p
#pragma xmp align a[i] with t[i]
...
#pragma acc parallel loop copy(a, sum) reduction(+:sum)
#pragma xmp loop on t[i] reduction(+:sum) acc
for(int i=0;i<N;i++){
  sum += a[i];
}
```

Fig. 6 Code example in XcalableMP extensions with OpenACC loop construct with reduction clause

and variable **sum** from host memory to accelerator memory. Moreover, in lines 7-8, the `parallel` directive and XcalableMP `loop` directive perform the next loop statement on accelerators in **nodes** in parallel. When finishing the calculation of the loop statement, OpenACC `reduction` clause and XcalableMP `reduction` and

`acc` clauses in lines 7-8 perform a reduction operation for the variable **sum** on accelerators in **nodes**.

2.3 Data Communication and Synchronization

2.3.1 `acc` clause

2.3.2 Coarray

2.4 Handling Multiple Accelerators

– devices Directive
  – on_device Clause
  – layout Clause
  – shadow Clause
  – barrier_device Construct

3 Omni XcalableACC Compiler

References

1. XcalableMP Language Specification, http://xcalablemp.org/specification.html (2017).
2. The OpenACC Application Programming Interface, http://www.openacc.org (2015).
3. MPI: A Message-Passing Interface Standard, http://mpi-forum.org (2015).