XcalableMP

⟨ex-scalable-em-p⟩

Langugae Specification

Version 1.0

XcalableMP Specification Working Group

November, 2011

Contents

List of Figures

Acknowledgment

The specification of XcalableMP is designed by the XcalableMP Specification Working Group, which consists of the following members from academia, research laboratories, and industries.

- Tatsuya Abe .... RIKEN
- Tokuro Anzaki .... Hitachi
- Taisuke Boku .... University of Tsukuba
- Toshio Endo .... TITECH
- Yasuharu Hayashi .... NEC
- Atsushi Hori .... RIKEN
- Kohichiro Hotta .... Fujitsu
- Hidetoshi Iwashita .... Fujitsu
- Jinpil Lee .... University of Tsukuba
- Yuichi Matsuo .... JAXA
- Kazuo Minami .... RIKEN
- Hitoshi Murai .... RIKEN
- Kengo Nakajima .... University of Tokyo
- Takashi Nakamura .... JAXA
- Tomotake Nakamura .... RIKEN
- Masahiro Nakao .... University of Tsukuba
- Takeshi Nanri .... Kyusyu University
- Kiyoshi Negishi .... Hitachi
- Yasuo Okabe .... Kyoto University
- Hitoshi Sakagami .... NIFS
- Shoich Sakon .... NEC
- Mitsuhisa Sato .... University of Tsukuba
- Takenori Shimosaka .... RIKEN
- Yoshihisa Shizawa .... RIST
- Hitoshi Uehara .... JAMSTEC
- Masahiro Yasugi .... Kyoto University
- Mitsuo Yokokawa .... RIKEN

Chapter 1

Intrinsic and Library Procedures

This specification defines various procedures for system inquiry, synchronization, computations, etc. The procedures are provided as intrinsic procedures in XcalableMP Fortran and library procedures in XcalableMP C.

1.1   System Inquiry Procedures

- `xmp_desc_of`

- `xmp_all_node_num`

- `xmp_all_num_nodes`

- `xmp_node_num`

- `xmp_num_nodes`

- `xmp_wtime`

- `xmp_wtick`

1.1.1   `xmp_desc_of`

Format

```
[F]   type(xmp_desc)   xmp_desc_of(xmp_entity)
[C]   xmp_desc_t       xmp_desc_of(xmp_entity)
```

Note that `xmp_desc_of` is an intrinsic function in XcalableMP Fortran or a built-in operator in XcalableMP C.

Synopsis

`xmp_desc_of` returns, in XcalableMP Fortran, or is evaluated to, in XcalableMP C, a descriptor to retrieve informations of the specified global array, template, or node array. The resulting descriptor can be used as an input argument of the inquiry procedures which is described in appendix C.

The type of the descriptor, `type(xmp_desc)`, in XcalableMP Fortran is implementation-dependent, and defined in a Fortran module named `xmp_lib` or a Fortran `include` file named `xmp_lib.h`.

The type of the descriptor, `xmp_desc_t`, in XcalableMP C is implementation-dependent, and defined in a header file named `xmp.h` in XcalableMP C.

Arguments

The argument or operand `xmp_entity` is the name of either a global array, a template or a node array.

1.1.2 `xmp_all_node_num`

Format

```
[F]   integer function   xmp_all_node_num()
[C]   int                 xmp_all_node_num(void)
```

Synopsis

The `xmp_all_node_num` routine returns the node number, within the primary node set, of the node that calls `xmp_all_node_num`.

Arguments

none.

1.1.3 `xmp_all_num_nodes`

Format

```
[F]   integer function   xmp_all_num_nodes()
[C]   int                 xmp_all_num_nodes(void)
```

Synopsis

The `xmp_all_num_nodes` routine returns the number of nodes in the entire node set.

Arguments

none.

1.1.4 `xmp_node_num`

Format

```
[F]   integer function   xmp_node_num()
[C]   int                 xmp_node_num(void)
```

Synopsis

The `xmp_node_num` routine returns the node number, within the current executing node set, of the node that calls `xmp_node_num`.

Arguments

none.

1.1.5 `xmp_num_nodes`

Format

```
[F]   integer function   xmp_num_nodes()
[C]   int                 xmp_num_nodes(void)
```

Synopsis

The `xmp_num_nodes` routine returns the number of the executing nodes.

Arguments

none.

1.1.6 `xmp_wtime`

Format

```
[F]   double precision function  xmp_wtime()
[C]   double                     xmp_wtime(void)
```

Synopsis

The `xmp_wtime` routine returns elapsed wall clock time in seconds since some time in the past. The "time in the past" is guaranteed not to change during the life of the process. There is no requirement that different nodes return "the same time."

Arguments

none.

1.1.7 `xmp_wtick`

Format

```
[F]   double precision function  xmp_wtick()
[C]   double                     xmp_wtick(void)
```

Synopsis

The `xmp_wtick` routine returns the resolution of the timer used by `xmp_wtime`. It returns a double precision value equal to the number of seconds between successive clock ticks.

Arguments

none.

1.2   Synchronization Procedures

1.2.1 `xmp_test_async`

```
[F]   logical function   xmp_test_async(async_id)
      integer            async_id

[C]   int                xmp_test_async(int async_id)
```

Synopsis

The `xmp_test_async` routine returns `.true.`, in XcalableMP Fortran, or `1`, in XcalableMP C, if an asynchronous communication specified by the argument `async_id` is complete; otherwise, it returns `.false.` or `0`.

Arguments

The argument `async_id` is an integer expression that specifies an asynchronous communication initiated by a global communication construct with the `async` clause.

1.3 Miscellaneous Procedures

1.3.1 xmp_gtol

```
[F]   integer function   xmp_gtol(d, g_idx, l_idx)
      type(xmp_desc)     d
      integer            g_idx(NDIMS)
      integer            l_idx(NDIMS)


[C]   void               xmp_gtol(xmp_desc_t d, int g_idx[], int l_idx[])
```

Synopsis

The `xmp_gtol` routine translates an index (specified by `g_idx`) of a global array (specified by `d`) into the corresponding index of its local section and sets to an array specified by `l_idx`. If the element of the specified index does not reside in the caller of the routine, the resulting array is set to an unspecified value.

Arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the target global array.

- [F] `g_idx` is a rank-one integer array of the size equal to the rank of the target global array specified by `d`.

- [F] `l_idx` is a rank-one integer array of the size equal to the rank of the target global array specified by `d`.

- [C] `g_idx` is a one-dimensional integer array.

- [C] `l_idx` is a one-dimensional integer array.

1.3.2 [C] xmp_malloc

```
void*   xmp_malloc(xmp_desc_t d, size_t size)
```

Synopsis

The `xmp_malloc` routine allocates a storage for the local section of a one-dimensional global array of size `size` that is associated with a descriptor specified by `d`, and returns the pointer to it on each node.

Arguments

- `d` is a descriptor, that is, an object of type `xmp_desc_t` that is associated with a pointer to the one-dimensional global array to be allocated.

- `size` is the size of the global array to be allocated.

Bibliography

[1] OpenMP Architecture Review Board, "OpenMP Application Program Interface Version 3.1", `http://www.openmp.org/mp-documents/OpenMP3.1.pdf` (2011).

[2] High Performance Fortran Forum, "High Performance Fortran Language Specification Version 2.0", `http://hpff.rice.edu/versions/hpf2/hpf-v20.pdf` (1997).

[3] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard Version 2.2", `http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf` (2009).

[4] Japan Association of High Performance Fortran, "HPF/JA Language Specification", `http://www.hpfpc.org/jahpf/spec/hpfja-v10-eng.pdf` (1999).

[5] Yuanyuan Zhang, Hidetoshi Iwashita, Kuninori Ishii, Masanori Kaneko, Tomotake Nakamura, and Kohichiro Hotta, "Hybrid Parallel Programming on SMP Clusters Using XPFortran and OpenMP", Proceedings of the International Workshop on OpenMP (IWOMP 2010), Vol. 6132 of Lecture Notes in Computer Science, pp. 133–148, Springer (2010).

[6] Hidetoshi Iwashita, Naoki Sueyasu, Sachio Kamiya, and Matthijs van Waveren, "VPP Fortran and the design of HPF/JA extensions", Concurrency and Computation — Practice & Experience, Vol. 14, No. 8–9, pp. 575–588, Wiley (2002).

[7] Jinpil Lee, Mitsuhisa Sato, and Taisuke Boku, "OpenMPD: A Directive-Based Data Parallel Language Extension for Distributed Memory Systems", Proceedings of the 2008 International Conference on Parallel Processing, pp. 121-128 (2008).

Appendix A

Programming Interface for MPI

This chapter describes the programming interface for MPI, which are widely used for parallel programming for cluster computing. Users can introduce MPI functions to XcalableMP using the interface.

XcalableMP provides the following user API functions to mix MPI functions with XcalableMP.

- `xmp_get_mpi_comm`

- `xmp_init_mpi`

- `xmp_finalize_mpi`

A.1   `xmp_get_mpi_comm`

Format

```
[F]   integer function   xmp_get_mpi_comm()
[C]   MPI_Comm           xmp_get_mpi_comm(void)
```

Synopsis

`xmp_get_mpi_comm` returns the handle of the communicator associated with the executing node set.

Arguments

none.

A.2   `xmp_init_mpi`

Format

```
[F]          xmp_init_mpi()
[C]   void   xmp_init_mpi(int *argc, char ***argv)
```

Synopsis

`xmp_init_mpi` initializes the MPI execution environment.

Arguments

In XcalableMP C, the command-line arguments `argc` and `argv` should be given to `xmp_init_mpi`.

A.3 `xmp_finalize_mpi`

Format

```
[F]        xmp_finalize_mpi()
[C]  void  xmp_finalize_mpi(void)
```

Synopsis

`xmp_finalize_mpi` terminates the MPI execution environment.

Arguments

none.

Example

```
────────────────── XcalableMP C ──────────────────
  #include <stdio.h>
  #include "mpi.h"
  #include "xmp.h"

5 #pragma xmp nodes p(4)

  int main(int argc, char *argv[]) {
    xmp_init_mpi(&argc, &argv)

10   int rank, size;
     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
     MPI_Comm_size(MPI_COMM_WORLD, &size);

  #pragma xmp task on p(2:3)
15 {
     MPI_Comm comm = xmp_get_mpi_comm(); // get the MPI communicator of p(2:3)

     int rank, size;
     MPI_Comm_rank(comm, &rank);
20   MPI_Comm_size(comm, &size);
  }

     xmp_finalize_mpi();

25   return 0;
  }
```

Appendix B

Directive for Thread Parallelism

Thread-level parallelism is needed to program multi-core cluster system. Users can use some features introduced from OpenMP to parallelize loops in thread level with the `threads` clause of the `loop` directive. No direct use of OpenMP directives in XcalableMP code is allowed.

B.1   threads clause

Syntax

[F]   `!$xmp loop` [(*loop-index* [, *loop-index*]...)]
                           `on` {*nodes-ref* | *template-ref*} [*reduction-clause*]... [*threads-clause*]
      do-loops

[C]   `#pragma xmp loop` [(*loop-index* [, *loop-index*]...)]
                           `on` {*nodes-ref* | *template-ref*} [*reduction-clause*]... [*threads-clause*]
      for-loops

   where *threads-clause* is:

      `threads` [*omp-clause*]

   and *omp-clause* is one of:

      `num_threads(` *num-thread* `)`
      `private(` *list* `)`
      `firstprivate(` *list* `)`
      `lastprivate(` *list* `)`

Description

OpenMP clauses such as `num_threads` can be specified in `threads` clause. The XcalableMP compiler generates internally OpenMP directives from the `loop` directive and the `threads` clause. Note that no `reduction` need to be specified in the `threads` clause because it is inherited from the `reduction` clause in the `loop` directive.

Example

This example calculates the total sum of an array. A `threads` clause is given to the `loop` directive to parallelize the loop statement in both process and thread level. The `reduction`

clause in the loop directive is also applied to the OpenMP directive which is generated by the XcalableMP compiler.

```
———————————————————— XcalableMP C ————————————————————
#include <stdio.h>
#include "xmp.h"
#define N 1024

#pragma xmp nodes p(*)
#pragma xmp template t(0:N-1)
#pragma xmp distribute t(block) onto p
#pragma xmp align a[i] with t(i)

int main(void) {
    . . . // initialize a[]

    int sum = 0;
#pragma xmp loop on t(i) reduction(+:sum) threads num_threads(4)
    for (int i = 0; i < N; i++) {
        sum += a[i];
    }

    return 0;
}
```

Appendix C

Interface to Numerical Libraries

This chapter describes the XcalableMP interfaces to existing MPI parallel libraries, which is effective to achieve high productivity and performance of XcalableMP programs.

C.1 Design of the Interface

A recommended design of the interface is as follows:

- Numerical library routines can be invoked by an XcalableMP procedure through an interface procedure (Figure C.1).
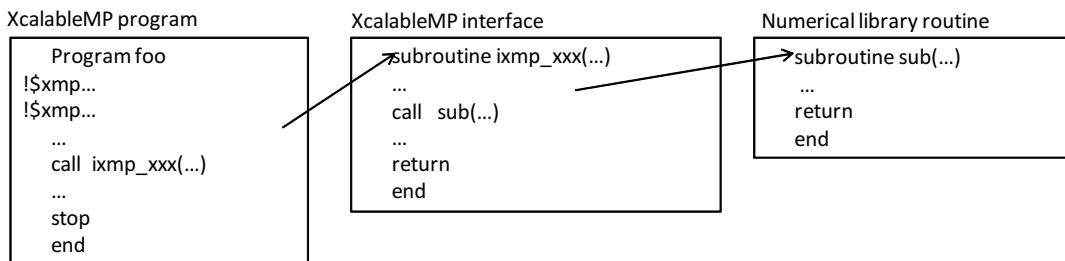

Figure C.1: Invocation of a Library Routine through an Interface Procedure

- When the numerical library routine needs information on an global array, the interface extracts it from the descriptor using some query routines provided by XcalableMP and passes it to the numerical library routine as arguments.

- The interface does not affect the behavior of numerical library routines except for restrictions concerning the XcalableMP specification.

C.2 Inquiry routines

All inquiry routines are specified by integer functions. When these inquiry routines return zero on normal completion, and other implementation-dependent integer on failure. Specifications of some inquiry routines are shown below.

C.2.1 xmp_nodes_ndims

Format

```
[F]   integer function   xmp_nodes_index(d, ndims)
      type(xmp_desc)      d
      integer             ndims
[C]   int                 xmp_nodes_ndims(xmp_desc_t d, int *ndims)
```

Synopsis

The `xmp_nodes_ndims` routine provides the rank of the target node array.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the node array.

Output arguments

- `ndims` is the rank of the node array.

C.2.2 xmp_nodes_index

Format

```
[F]   integer function   xmp_nodes_index(d, index)
      type(xmp_desc)      d
      integer             dim
      integer             index
[C]   int                 xmp_nodes_index(xmp_desc_t d, int dim, int *index)
```

Synopsis

The `xmp_nodes_index` routine provides the indices of the executing node in the target node array.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)` in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the node array.

- `dim` is a target dimension of the node array.

Output arguments

- `index` is the index of the target dimension of the node array.

C.2.3 xmp_nodes_size

Format

```
[F]   integer function   xmp_nodes_size(d, dim, size)
      type(xmp_desc)      d
      integer             dim
      integer             size
[C]   int                 xmp_nodes_size(xmp_desc_t d, int dim, int *size)
```

Synopsis

The `xmp_nodes_size` routine provides the size of each dimension of the target node array.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the node array.

- `dim` is a target dimension of the node array.

Output arguments

- `size` is a extent of the target dimension of the node array.

C.2.4 `xmp_nodes_attr`

Format

```
[F]   integer function  xmp_nodes_attr(d, attr)
      type(xmp_desc)    d
      integer           attr
[C]   int               xmp_nodes_attr(xmp_desc_t d, int *attr)
```

Synopsis

The `xmp_nodes_attr` routine provides the attribute of the target node array. This routine outputs one of:

`XMP_ENTIRE_NODES`	(Entire nodes)
`XMP_EXECUTING_NODES`	(Executing nodes)
`XMP_PRIMARY_NODES`	(Primary nodes)
`XMP_EQUIVALENCE_NODES`	(Equivalence nodes)

Each output integer value for the attribute is implementation-dependent.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the node array.

Output arguments

- `attr` is the attribute of the target node array.

C.2.5 `xmp_template_fixed`

Format

```
[F]   integer function  xmp_template_fixed(d, fixed)
      type(xmp_desc)    d
      logical           fixed
[C]   int               xmp_template_fixed(xmp_desc_t d, _Bool *fixed)
```

Synopsis

The `xmp_template_fixed` routine provides the logical value which shows whether the template is fixed or not.

Input arguments

- d is a descriptor, that is, an object of type type(xmp_desc), in XcalableMP Fortran, or xmp_desc_t, in XcalableMP C, that is associated with the template.

Output arguments

- fixed is a logical scalar. When the template is fixed, fixed is true.

C.2.6 xmp_template_ndims

Format

```
[F]   integer function   xmp_template_ndims(d, ndims)
      type(xmp_desc)     d
      integer            ndims
[C]   int                xmp_template_ndims(xmp_desc_t d, int *ndims)
```

Synopsis

The xmp_template_ndims routine provides the rank of the target template

Input arguments

- d is a descriptor, that is, an object of type type(xmp_desc), in XcalableMP Fortran, or xmp_desc_t, in XcalableMP C, that is associated with the template.

Output arguments

- ndims is the rank of the template.

C.2.7 xmp_template_lbound

Format

```
[F]   integer function   xmp_template_lbound(d, dim, lbound)
      type(xmp_desc)     d
      integer            dim
      integer            lbound
[C]   int                xmp_template_lbound(xmp_desc_t d, int dim, int *lbound)
```

Synopsis

The xmp_template_lbound routine provides the lower bound of each dimension of the template. When the lower bound is not fixed, this routine the implemetation-dependent negative integer value that indicates XMP_UNDIFINED.

Input arguments

- d is a descriptor, that is, an object of type type(xmp_desc), in XcalableMP Fortran, or xmp_desc_t, in XcalableMP C, that is associated with the template.

- dim is a target dimension of the template.

Output arguments

- `lbound` is lower bound of target dimension of the template.

C.2.8 `xmp_template_ubound`

Format

```
[F]   integer function   xmp_template_ubound(d, dim, ubound)
      type(xmp_desc)      d
      integer             dim
      integer             ubound
[C]   int                 xmp_template_ubound(xmp_desc_t d, int dim, int *ubound)
```

Synopsis

The `xmp_template_ubound` routine provides the upper bound of each dimension of the template. When the upper bound is not fixed, this routine returns `XMP_UNDIFINED`.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the template.

- `dim` is a target dimension of the template.

Output arguments

- `ubound` is upper bound of target dimension of the template.

C.2.9 `xmp_dist_format`

Format

```
[F]   integer function   xmp_dist_format(d, dim, format)
      type(xmp_desc)      d
      integer             dim
      integer             format
[C]   int                 xmp_dist_format(xmp_desc_t d, int dim, int *format)
```

Synopsis

The `xmp_dist_format` routine provides the distribution format of each dimension of the target template. This routine outputs one of:

`XMP_NOT_DISTRIBUTED`	(Not distributed)
`XMP_BLOCK`	(Block distribution)
`XMP_CYCLIC`	(Cyclic distribution)
`XMP_GBLOCK`	(Gblock distribution)

Each output integer value for the distribution format is implementation-dependent.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the template.

- `dim` is a target dimension of the template.

Output arguments

- `format` is the distribution format of target dimension of the template.

C.2.10   `xmp_dist_blocksize`

Format

```
[F]  integer function  xmp_dist_blocksize(d, dim, blocksize)
     type(xmp_desc)    d
     integer           dim
     integer           blocksize
[C]  int               xmp_dist_blocksize(xmp_desc_t d, int dim, int *blocksize)
```

Synopsis

The `xmp_dist_blocksize` routine provides the blocking extent of each dimension of the target template.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the template.

- `dim` is a target dimension of the template.

Output arguments

- `blocksize` is the blocking extent of the target dimension of the template.

C.2.11   `xmp_dist_gblockmap`

Format

```
[F]  integer function  xmp_dist_gblockmap(d, dim, n, map)
     type(xmp_desc)    d
     integer           dim
     integer           n
     integer           map(n)
[C]  int               xmp_dist_gblockmap(xmp_desc_t d, int dim, int n, int *map[])
```

Synopsis

The `xmp_dist_gblockmap` routine provides a mapping array. When dim dimension of the global array is distributed by gblock and a mapping array is fixed, this routine returns zero. When a mapping array is not fixed, this routine returns `XMP_UNDIFINED`.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the template.

- `dim` is a target dimension of the template.

- `n` is an extent of the mapping array. n must be more than the size of the corresponding dimension of the node array that the template maps.

Output arguments

- map is a one-dimensional integer array. The i-th element of map is the value of the i-th element of the target mapping array.

C.2.12 xmp_dist_nodes

Format

```
[F]  integer function  xmp_dist_nodes(d, dn)
     type(xmp_desc)    d
     type(xmp_desc)    dn
[C]  int               xmp_dist_nodes(xmp_desc_t d, xmp_desc_t *dn)
```

Synopsis

The xmp_dist_nodes routine provides an object which of type type(xmp_desc) is associated with the node array.

Input arguments

- d is a descriptor, that is, an object of type type(xmp_desc), in XcalableMP Fortran, or xmp_desc_t, in XcalableMP C, that is associated with the template.

Output arguments

- dn is a descriptor, that is, an object of type type(xmp_desc), in XcalableMP Fortran, or xmp_desc_t, in XcalableMP C, that is associated with the node array.

C.2.13 xmp_dist_axis

Format

```
[F]  integer function  xmp_dist_axis(d, dim, axis)
     type(xmp_desc)    d
     integer           dim
     integer           axis
[C]  int               xmp_dist_axis(xmp_desc_t d, int dim, int *axis)
```

Synopsis

The xmp_dist_axis routine provides the dimension of the node array associated with each dimension of the template. When no dimension of a node array maps a dimension of a template, this routine returns XMP_UNDIFINED.

Input arguments

- d is a descriptor, that is, an object of type type(xmp_desc), in XcalableMP Fortran, or xmp_desc_t, in XcalableMP C, that is associated with the template.

- dim is a target dimension of the template.

Output arguments

- axis is the dimension of the node array associated with target dimension of the template.

C.2.14 `xmp_align_axis`

Format

```
[F]   integer function   xmp_align_axis(d, dim, axis)
      type(xmp_desc)      d
      integer             dim
      integer             axis
[C]   int                 xmp_align_axis(xmp_desc_t d, int dim, int *axis)
```

Synopsis

The `xmp_dist_axis` routine provides the dimension of the template associated with each dimension of the global array. When no dimension of a template maps a dimension of a global array, this routine returns `XMP_UNDIFINED`.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the global array.

- `dim` is a target dimension of the global array.

Output arguments

- `axis` is the dimension of the template associated with target dimension of the global array.

C.2.15 `xmp_align_offset`

Format

```
[F]   integer function   xmp_align_offset(d, dim, offset)
      type(xmp_desc)      d
      integer             dim
      integer             offset
[C]   int                 xmp_align_offset(xmp_desc_t d, int dim, int *offset)
```

Synopsis

The `xmp_align_offset` routine provides the offset of each dimension of the target global array. When there is no offset, this routine returns `XMP_UNDIFINED`.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the global array.

- `dim` is a target dimension of the global array.

Output arguments

- `offset` is the offset of target dimension of the global array.

C.2.16 xmp_align_replicated

Format

```
[F]   integer function   xmp_align_replicated(d, dim, replicated)
      type(xmp_desc)      d
      integer             dim
      integer             replicated
[C]   int                 xmp_align_replicated(xmp_desc_t d, int dim, int *replicated)
```

Synopsis

The `xmp_align_replicated` routine provides the logical value which shows whether the target dimension of the template that aligns the global array is replicated or not. When the dimension of the template is replicated, this routine returns true.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the global array.

- `dim` is a target dimension of the global array.

Output arguments

- `replicated` is a logical scalar.

C.2.17 xmp_align_template

Format

```
[F]   integer function   xmp_align_template(d, dt)
      type(xmp_desc)      d
      type(xmp_desc)      dt
[C]   int                 xmp_align_template(xmp_desc_t d, xmp_desc_t *dn)
```

Synopsis

The `xmp_align_template` routine provides an object which of type `type(xmp_desc)` is associated with the template.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the global array.

Output arguments

- `dt` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the template.

C.2.18 `xmp_array_ndims`

Format

```
[F]   integer function   xmp_array_ndims(d, ndims)
      type(xmp_desc)      d
      integer             ndims
[C]   int                 xmp_array_ndims(xmp_desc_t d, int *ndims)
```

Synopsis

The `xmp_array_ndims` routine provides the rank of the target global array

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the global array.

Output arguments

- `ndims` is the rank of the global array.

C.2.19 `xmp_array_lshadow`

Format

```
[F]   integer function   xmp_array_lshadow(d, dim, lshadow)
      type(xmp_desc)      d
      integer             dim
      integer             lshadow
[C]   int                 xmp_array_lshadow(xmp_desc_t d, int dim, int *lshadow)
```

Synopsis

The `xmp_array_lshadow` routine provides the size of lower shadow of each dimension of the target global array.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or `xmp_desc_t`, in XcalableMP C, that is associated with the global array.

- `dim` is a target dimension of the global array.

Output arguments

- `lshadow` is the size of lower shadow of target dimension of the global array.

C.2.20 `xmp_array_ushadow`

Format

```
[F]   integer function   xmp_array_ushadow(d, dim, ushadow)
      type(xmp_desc)      d
      integer             dim
      integer             ushadow
[C]   int                 xmp_array_ushadow(xmp_desc_t d, int dim, int *ushadow)
```

Synopsis

The `xmp_array_ushadow` routine provides the size of upper shadow of each dimension of the
target global array.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or
  `xmp_desc_t`, in XcalableMP C, that is associated with the global array.

- `dim` is a target dimension of the global array.

Output arguments

- `ushadow` is the size of upper shadow of target dimension of the global array.

C.2.21   `xmp_array_lbound`

Format

```
[F]   integer function   xmp_array_lbound(d, dim, lbound)
      type(xmp_desc)      d
      integer             dim
      integer             lbound
[C]   int                 xmp_array_lbound(xmp_desc_t d, int dim, int *lbound)
```

Synopsis

The `xmp_array_lbound` routine provides the lower bound of each dimension of the global array.
When the lower bound is not fixed, this routine returns `XMP_UNDIFINED`.

Input arguments

- `d` is a descriptor, that is, an object of type `type(xmp_desc)`, in XcalableMP Fortran, or
  `xmp_desc_t`, in XcalableMP C, that is associated with the global array.

- `dim` is a target dimension of the global array.

Output arguments

- `lbound` is lower bound of target dimension of the global array.

C.2.22   `xmp_array_ubound`

Format

```
[F]   integer function   xmp_array_ubound(d, dim, ubound)
      type(xmp_desc)      d
      integer             dim
      integer             ubound
[C]   int                 xmp_array_ubound(xmp_desc_t d, int dim, int *ubound)
```

Synopsis

The `xmp_array_ubound` routine provides the upper bound of each dimension of the global array.
When the upper bound is not fixed, this routine returns `XMP_UNDIFINED`.

Input arguments

- **d** is a descriptor, that is, an object of type **type(xmp desc)**, in XcalableMP Fortran, or **xmp desc t**, in XcalableMP C, that is associated with the global array.

- **dim** is a target dimension of the global array.

Output arguments

- **ubound** is upper bound of target dimension of the global array.

C.3 Inquiry routines(Implementation-dependent)

In this section, implementation-dependent functions are shown. Specifications of inquiry functions below are samples of Omni XcalableMP Compiler.

C.3.1 xmp_array_lsize

Format

```
[F]   integer function   xmp_array_lsize(d, dim, lsize)
      type(xmp_desc)     d
      integer            dim
      integer            lsize
[C]   int                xmp_array_lsize(xmp_desc_t d, int dim, int lsize)
```

Synopsis

The **xmp array lsize** routine provides the local size of each dimension of the target global array. When the target global array has a shadow, the local size include the size of the shadow. Each output value is implementation-dependent.

Input arguments

- **d** is a descriptor, that is, an object of type **type(xmp desc)**, in XcalableMP Fortran, or **xmp desc t**, in XcalableMP C, that is associated with the global array.

- **dim** is a target dimension of the global array.

Output arguments

- **lsize** is the local size of target dimension of the global array.

C.3.2 xmp_array_laddr

Format

```
[C]   int   xmp_array_laddr(xmp_desc_t d, void **laddr)
```

Synopsis

The **xmp array laddr** routine provides the local address of the target global array. The local address is implementation-dependent.

Input arguments

- d is a descriptor, that is, an object of type xmp_desc_t, in XcalableMP C, that is associated with the global array.

Output arguments

- laddr is the local address of the target global array.

C.3.3   xmp_array_lead_dim

Format

```
[F]   integer function   xmp_array_lead_dim(d, n, size)
      type(xmp_desc)     d
      integer            n
      integer            size(n)
[C]   int                xmp_array_lead_dim(xmp_desc_t d, int n, int *size[])
```

Synopsis

The xmp_array_lead_dim routine provides the leading dimension of each local section of the target global array. Each output value is implementation-dependent.

Input arguments

- d is a descriptor, that is, an object of type type(xmp_desc), in XcalableMP Fortran, or xmp_desc_t, in XcalableMP C, that is associated with the global array.

- n is an extent of one-dimensional array size. n must be more than ndims - 1. ndims is the rank of the global array.

Output arguments

- size is a one-dimensional integer array.

C.4   Example

This section shows the interface to ScaLAPACK as an example of the XcalableMP interface to numerical libraries.

ScaLAPACK is a linear algebra library for distributed-memory. Communication processes in the ScaLAPACK routines depends on BLACS (Basic Linear Algebraic Communication Subprograms). ScaLAPACK library routines invoked from XcalableMP procedures also depend on BLACS.

Example 1 This example shows an implementation of the interface for the ScaLAPACK driver routine pdgesv.

```
                      XcalableMP Fortran
      subroutine ixmp_pdgesv(n,nrhs,a,ia,ja,da,ipiv,b,ib,jb,db,ictxt,info)

      use xmp_lib

5     integer n,nrhs,ia,ja,ib,jb,ictxt,info,desca(9),descb(9),ierr
```

```
        double precision a,b
        type(xmp_desc) da,db,dta,dtb
        integer lbound_a1,ubound_a1,lbound_a2,ubound_a2
        integer blocksize_a1,blocksize_a2,lead_dim_a
10      integer lbound_b1,ubound_b1,lbound_b2,ubound_b2
        integer blocksize_b1,blocksize_b2,lead_dim_b

        ierr=xmp_array_lbound(da,1,lbound_a1)
        ierr=xmp_array_ubound(da,1,ubound_a1)
15      ierr=xmp_array_lbound(da,2,lbound_a2)
        ierr=xmp_array_ubound(da,2,ubound_a2)
        ierr=xmp_align_template(da,dta)
        ierr=xmp_dist_blocksize(dta,1,blocksize_a1)
        ierr=xmp_dist_blocksize(dta,2,blocksize_a2)
20      ierr=xmp_array_lead_dim(da,1,lead_dim_a)

        ierr=xmp_array_lbound(db,1,lbound_b1)
        ierr=xmp_array_ubound(db,1,ubound_b1)
        ierr=xmp_array_lbound(db,2,lbound_b2)
25      ierr=xmp_array_ubound(db,2,ubound_b2)
        ierr=xmp_align_template(db,dtb)
        ierr=xmp_dist_blocksize(dtb,1,blocksize_b1)
        ierr=xmp_dist_blocksize(dtb,2,blocksize_b2)
        ierr=xmp_array_lead_dim(db,1,lead_dim_b)
30
        desca(1)=1
        desca(2)=ictxt
        desca(3)=ubound_a1-lbound_a1+1
        desca(4)=ubound_a2-lbound_a2+1
35      desca(5)=blocksize_a1
        desca(6)=blocksize_a2
        desca(7)=0
        desca(8)=0
        desca(9)=lead_dim_a
40
        descb(1)=1
        descb(2)=ictxt
        descb(3)=ubound_b1-lbound_b1+1
        descb(4)=ubound_b2-lbound_b2+1
45      descb(5)=blocksize_b1
        descb(6)=blocksize_b2
        descb(7)=0
        descb(8)=0
        descb(9)=lead_dim_b
50
        call pdgesv(n,nhrs,a,ia,ja,desca,ipiv,b,ib,jb,descb,info)

        return
        end
```

Example 2 This example shows an XcalableMP procedure using the interface of Example 1.

```
―――――――――――― XcalableMP Fortran ――――――――――――
        program xmptdgesv

        use xmp_lib

5       double precision a(1000,1000)
        double precision b(1000)
        integer ipiv(2*1000,2)
!$xmp nodes p(2,2)
!$xmp template t(1000,1000)
10 !$xmp template t1(2*1000,2)
!$xmp distribute t(block,block) onto p
!$xmp distribute t1(block,block) onto p
!$xmp align a(i,j) with t(i,j)
!$xmp align ipiv(i,j) with t1(i,j)
15 !$xmp align b(i) with t(i,*)
        ...
        integer i,j,ictxt
        integer m=1000,n=1000,nprow=2,npcol=2
        integer icontxt=-1,iwhat=0
20      integer nrhs=1,ia=1,ja=1,ib=1,jb=1,info
        character*1 order
        ...
        order="C"
        ...
25      call blacs_get(icontxt,iwhat,ictxt)
        call blacs_gridinit(ictxt,order,nprow,npcol)
        ...
!$xmp loop (i,j) on t(i,j)
        do j=1,n
30         do i=1,m
              a(i,j) = ...
           end do
        end do
        ...
35 !$xmp loop on t(i,*)
        do i=1,m
           b(i)= ...
        end do
        ...
40      call ixmp_pdgesv(n,nrhs,a,ia,ja,xmp_desc_of(a),ipiv,
       *              b,ib,jb,xmp_desc_of(b),ictxt,info)
        ...
        call blacs_gridexit(ictxt)
        ...
45      stop
```

```
end
```