Table of Contents

Omni Compiler Guidebook

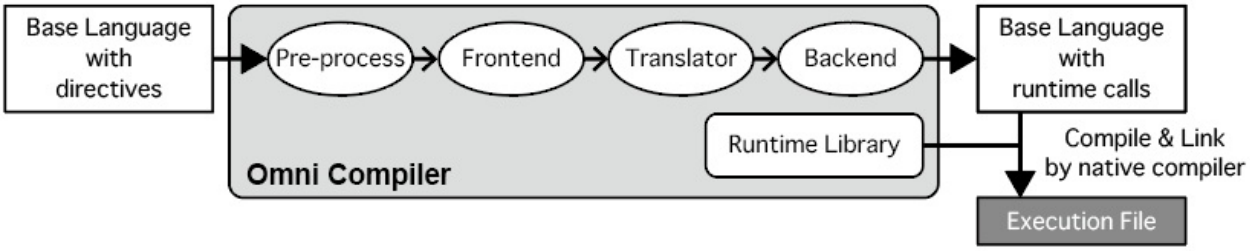
This is a guidebook for anyone who wants to learn how to install the omni compiler, how to use it, and some tips.

There are two versions of the omni compiler, **Stable version** and **Nightly build version**. The Stable version is the so-called official version, and the Nightly build version is a trial version that is released at midnight on our official website. In this guidebook, we usually describe information about the latest Stable version (version 1.0.1).

What is the omni compiler ?

The omni compiler is a compiler for code including XcalableMP, XcalableACC, and OpenACC directives. The base languages supported by the omni compiler are C language (C99) and Fortran 2003 in XcalableMP, and C language (C99) in XcalableACC and OpenACC.

The omni compiler is one of the translator compilers that translate from code including directives to code including runtime calls. In the omni compiler, XcodeML is used to analyze code in an intermediate code format of XML expression. The following figure shows the operation flow of the omni compiler.



The omni compiler uses a native compiler, for example `mpicc` or `nvcc`, to create an execution file from translated code including runtime calls. The runtime library provided by the omni compiler uses MPI in XcalableMP, and CUDA in OpenACC, and both MPI and CUDA in XcalableACC.

In XcalableMP, the omni compiler may create better runtime libraries by adding one of the onesided communication libraries to MPI. We describe these in detail in 2.3. Optional instructions.

The omni compiler is developed by the following two groups.

- Programming Environment Research Team, AICS, RIKEN
- HPCS Laboratory, Graduate School of Systems & Information Engineering, University of Tsukuba

How to install

Introduction

You can install the omni compiler by using the general installation method on UNIX (`./configure; make; make install`).

When executing `./configure` without options, only XcalableMP is installed. If you want to install XcalableACC and OpenACC, you need to add options to `./configure` . This section, 2.1. General instructions, and Each supercomputer explain how to install XcalableMP. 2.3. Optional instructions explains how to install XcalableACC and OpenACC.

Software dependencies

Before you start to install the omni compiler, the following software must be installed.

- Yacc
- Lex
- C Compiler (supports C99)
- Fortran Compiler (supports Fortran 90)
- C++ Compiler
- Java Compiler
- MPI Implementation (supports MPI-2 or over)
- libxml2
- make

The following shows the procedure for installing software in major Linux distributions.

- Debian GNU/Linux 8.3

```
# aptitude install flex gcc gfortran g++ openjdk-7-jdk libopenmpi-dev openmpi-bin \
  libxml2-dev byacc make perl
```

- Ubuntu 15.10

```
$ sudo apt-get install flex gcc gfortran g++ openjdk-7-jdk libopenmpi-dev openmpi-bin \
  libxml2-dev byacc make perl
```

- CentOS 7.2

```
# yum install flex gcc gfortran gcc-c++ java-1.7.0-openjdk-devel openmpi-devel \
  libxml2-devel byacc make perl
```

General instructions

This section explains how to install the omni compiler in a general Unix environment.

Build and Install

```
$ ./configure --prefix=(INSTALL PATH)
$ make
$ make install
```

`(INSTALL PATH)` indicates the place where the omni compiler is installed.

Set PATH

- bash and zsh

  ```
  $ export PATH=(INSTALL PATH)/bin:$PATH
  ```

- csh and tcsh

  ```
  % setenv PATH (INSTALL PATH)/bin:$PATH
  ```

Each supercomputers

When you add an option `--target=(machine name)` to `./configure`, you can build the omni compiler that is suitable for the following specific architectures.

- The K computer
- Fujitsu FX100
- Fujitsu FX10
- NEC SX-ACE
- NEC SX9
- HITACHI SR16000
- IBM BlueGene/Q

The K computer

```
$ ./configure --target=Kcomputer-linux-gnu --prefix=(INSTALL PATH)
$ make
$ make install
```

Fujitsu FX100

```
$ ./configure --target=FX100-linux-gnu --prefix=(INSTALL PATH)
$ make
$ make install
```

Fujitsu FX10

```
$ ./configure --target=FX10-linux-gnu --prefix=(INSTALL PATH)
$ make
$ make install
```

NEC SX-ACE

If a login node does not have **"libxml2"**, you need to install **"libxml2"** from its official website.

```
$ tar xfz libxml2-git-snapshot.tar.gz
$ cd libxml2-2.9.2
$ ./configure --without-python --prefix=(LIBXML2 PATH)
$ make
$ make install
```

Next, you install the omni compiler.

```
$ ./configure --target=sxace-nec-superux --with-libxml2=(LIBXML2 PATH) --prefix=(INSTALL PATH)
$ make
$ make install
```

NEC SX9

```
$ ./configure --target=sx9-nec-superux --prefix=(INSTALL PATH)
$ make
$ make install
```

HITACHI SR16000

```
$ bash
$ export PATH=/opt/freeware/bin/:$PATH
$ export PATH=/usr/java6/jre/bin/:$PATH
$ bash ./configure --target=powerpc-hitachi-aix --prefix=(INSTALL PATH)
$ make
$ make install
```

IBM BlueGene/Q

If a login node does not have **"Java"**, you need to install **"Java"**. For example, you can get "openjdk1.7.0-ppc-aix-port-linux-ppc64-b**.tar.bz2" from the OpenJDK website.

```
$ ./configure --target=powerpc-ibm-cnk --prefix=(INSTALL PATH)
$ make
$ make install
```

Optional instructions

How to install OpenACC

You add `--enable-openacc` to `./configure`. If you need, you also add install PATH of cuda by `--with-cuda=(CUDA PATH)`.

```
$ ./configure --enable-openacc --with-cuda=(CUDA PATH)
$ make
$ make install
```

It may be possible to generate a more suitable runtime library by setting options for the `nvcc` command, which is used to generate the runtime library for OpenACC. In that case, you can add the `--with-gpu-cflags="(NVCC CFLAGS)"` option to `./configure`.

```
$ ./configure --enable-openacc --with-cuda=(CUDA PATH) --with-gpu-cflags="-arch=sm_20 -O3"
```

How to install XcalableACC

You add `--enable-openacc --enable-xacc` to `./configure`. As with OpenACC, you can add the `--with-cuda=(CUDA PATH)` and `--with-gpu-cflags="(NVCC CFLAGS)"` options to `./configure`.

```
$ ./configure --enable-openacc --enable-xacc --with-cuda=(CUDA PATH)
$ make
$ make install
```

Use of onesided library on XcalableMP

You may generate a better runtime library by using MPI and a onesided library on XcalableMP. The omni compiler supports the following onesided libraries.

- Fujitsu MPI Extended RDMA
- GASNet
- MPI Version 3

Fujitsu MPI Extended RDMA

Fujitsu MPI Extended RDMA is available only on the K computer, FX100, and FX10. By using `./configure --target=(machine name)` explained in 2.2. Each supercomputer, the omni compiler automatically uses Fujitsu MPI Extended RDMA.

GASNet

GASNet is a onesided communication library developed at U.C. Berkeley. If you want to use GASNet, you should add **"install path of GASNet"** and **"its conduit"** to `./configure`.

```
$ ./configure --with-gasnet=(GASNET PATH) --with-gasnet-conduit=(GASNET CONDUIT)
```

When you omit `--with-gasnet-conduit=(GASNET CONDUIT)`, the omni compiler automatically selects an available conduit.

MPI Version 3

The omni compiler automatically selects MPI Version 3 under the following conditions.

- Using MPI implementation supports MPI Version 3
- Not using GASNet
- Except for the K computer, FX100, and FX10

How to confirm onesided library

You can confirm which onesided communication library the omni compiler used in the last output of `./configure`.

- Fujitsu MPI Extended RDMA

```
Onesided                    : yes
Communication Library       : Fujitsu RDMA
```

- GASNet

```
Onesided                    : yes
Communication Library       : GASNet
```

- MPI Version 3

```
Onesided                    : yes
Communication Library       : MPI3
```

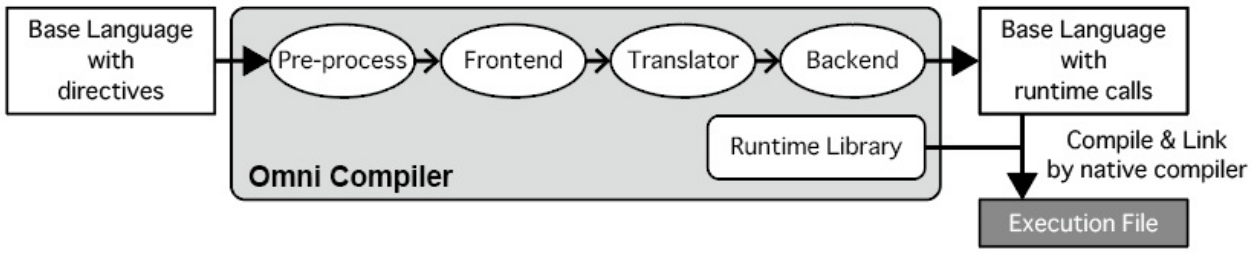
- Not use onesided library

```
Onesided                    : no
```

How to indicate compiler used by the omni compiler

The compiler used by the omni compiler can be classified into two types by the location of its binary.

- **Local compiler** is used in the Pre-process, Frontend, Translator, and Backend processes. A binary generated by a local compiler is used on the machine where you build the omni compiler, for example, the login node of a cluster system.
- **Native compiler** is used to generate an execution file and runtime library of the omni compiler. A binary generated by a native compiler is used on the machine where you carry out calculations, for example, the compute node of a cluster system.



Even though the omni compiler automatically selects the above compilers when executing `./configure`, you can select them by using the following variables.

- Local compiler

Variable	Description
CC	C compiler
CFLAGS	C compiler flags
FC	Fortran compiler
FCFLAGS	Fortran compiler flags
JAVA	Java application launcher
JAVAC	Java compiler
JAR	Java Archive Tool

- Native compiler

Variable	Description
MPI_CPP	C preprocessor
MPI_CPPFLAGS	C preprocessor flags
MPI_CC	C compiler
MPI_CFLAGS	C compiler flags
MPI_CLIBS	C compiler linker flags
MPI_FPP	Fortran preprocessor
MPI_FPPFLAGS	Fortran preprocessor flags
MPI_FC	Fortran compiler
MPI_FCFLAGS	Fortran compiler flags
MPI_FCLIBS	Fortran compiler linker flags

For example, if you want to use the `icc` for `CC`, you execute `./configure CC=icc`.

Use of BLAS for runtime library

Part of the runtime library of the omni compiler can use BLAS. For example, when a function `xmp_matmul()` that is one of the intrinsic functions uses BLAS, it may execute faster.

Not select (Default)

Internal functions prepared in the runtime library are used.

The K computer

When executing `./configure --target=Kcomputer-linux-gnu`, the runtime library uses BLAS provided in the K computer.

FX100 or FX10

When executing `./configure --enable-SSL2BLAMP`, the runtime library uses BLAS provided in FX100 or FX10.

Intel MKL

When executing `./configure --enable-intelmkl`, the runtime library uses Intel MKL.

Selected BLAS

When executing `./configure --with-libblas=(BLAS PATH)`, the runtime library uses its BLAS.

How to use

This section describes how to compile a code where XcalableMP, XcalableACC, and OpenACC directives exist, and how to execute binary of it.

How to compile

Examples of compile command

XcalableMP/C

```
$ xmpcc a.c
```

XcalableMP/Fortran

```
$ xmpf90 a.f90
```

XcalableACC/C

```
$ xmpcc -xacc a.c
```

OpenACC/C

```
$ ompcc -acc a.c
```

About the compile option

As stated in 1. What is the omni compiler, the native compiler finally compiles the code translated by the omni compiler, so all compile options except for options specific to the omni compiler are passed to the native compiler. For example, when using the optimization option `-O2` that is often used, `-O2` is passed to the native compiler.

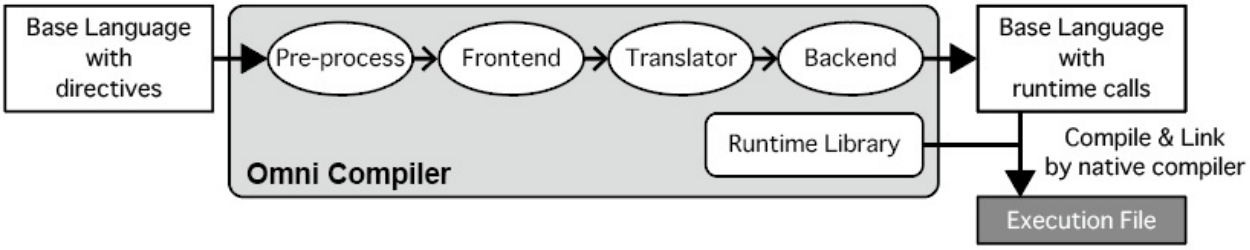
```
$ xmpcc -O2 a.c
```

The omni compiler specific options

Classification of options

The omni compiler specific option is classified into a **"common option"** and a **"unique option."** Moreover, the common option is classified into a **"compile driver option"** and a **"process option."**

- Common option is an option for all languages and directives.
  - Compile driver option is an option applied to all compilation processes.
  - Process option is an option applied to each compilation process.
- Unique option is an option for each language and directive.



Common options

- Compile driver option

Option	Description
-o <file>	place the output into
-I <dir>	add the directory dir to the list of directories to be searched for header files
-c	compile and assemble, but do not link
-E	preprocess only; do not compile, assemble or link
-v,--verbose	print processing status
--version	print version
-h,--help	print usage
--show-env	show environment variables
--tmp	output translated code to __omni_tmp__<file>
--dry	only print processing status (not compile)
--debug	save intermediate files to ./__omni_tmp__/
--stop-pp	save intermediate files and stop after preprocess
--stop-frontend	save intermediate files and stop after frontend
--stop-translator	save intermediate files and stop after translator
--stop-backend	save intermediate files and stop after backend
--stop-compile	save intermediate files and stop after compile

- Process option

Option	Description
--Wp[option]	add preprocessor option
--Wf[option]	add frontend option
--Wx[option]	add Xcode translator option
--Wb[option]	add backend option
--Wn[option]	add native compiler option
--Wl[option]	add linker option

For example, if you want to add the option `-Ltest` to only a linker process, you execute it as follows.

```
$ xmpcc --Wl"-Ltest" a.c
```

Unique option

- XcalableMP/C

Option	Description
-omp, --openmp	enable OpenMP function
--profile scalasca	output results in scalasca format for all directives
--profile tlog	output results in tlog format for all directives
--selective-profile scalasca	output results in scalasca format for selected directives
--selective-profile tlog	output results in tlog format for selected directives

- XcalableMP/Fortran

Option	Description
-omp, --openmp	enable OpenMP function
-J <dir>	specify where to put .mod and .xmod files for compiled modules
-cpp	enable preprocess
-max_assumed_shape=N	specifies maximum assumed-shape array arguments (default: 16)

- XcalableACC/C

Option	Description
-xacc, --xcalableacc	enable XcalableACC function
--no-ldg	disable use of read-only data cache
--default-veclen=LENGTH	specify default vector length (default: 256)

- OpenACC/C

Option	Description
-acc, --openacc	enable OpenACC function
--no-ldg	disable use of read-only data cache
--default-veclen=LENGTH	specify default vector length (default: 256)

How to execute

XcalableMP and XcalableACC

Because the runtime libraries of XcalableMP and XcalableACC use MPI, you execute a program by using an MPI execution command, for example, the `mpiexec` or `mpirun` command. Except when the runtime library uses GASNet described in "Use of onesided library on XcalableMP" in 2.3. Optional instructions, you execute a program by using the GASNet execution command (`gasnetrun_XXX` . `XXX` is a conduit name).

- Not using GASNet

```
$ mpiexec -n 2 ./a.out
```

- Using GASNet with ibv-conduit

```
$ gasnetrun_ibv -n 2 ./a.out
```

OpenACC

You can execute a program by using the general instructions.

```
$ ./a.out
```

Environmental variables

XcalableMP and XcalableACC

XMP_NODE_SIZEn

In the XcalableMP specification, `*` is available in the last dimension of the definition of a node set.

- C language

```
#pragma xmp nodes p(2,*)
```

- Fortran

```
!$xmp nodes p(2,*)
```

The omni compiler extends the XcalableMP specification to use `*` except for the last dimension.

- C language

```
#pragma xmp nodes p(*,*)
```

- Fortran

```
!$xmp nodes p(*,*)
```

The valule of the n'th dimension node set is set by using the environmental valiable `XMP_NODE_SIZEn`. The `n` is the 0-origin integer number. For example, assume `XMP_NODE_SIZE0` and `XMP_NODE_SIZE1` are set as follows.

```
$ export XMP_NODE_SIZE0=2
$ export XMP_NODE_SIZE1=4
$ mpirun -np 8 ./a.out
```

The above example code is the same as follows.

- C language

```
#pragma xmp nodes p(2,4)
```

- Fortran

```
!$xmp nodes p(2,4)
```

XMP_ONESIDED_HEAP_SIZE (Only GASNet and MPI version 3)

The `XMP_ONESIDED_HEAP_SIZE` must be set when the following execution error occurs.

```
[ERROR] Cannot allocate coarray. Heap memory size of coarray is too small.
        Please set the environmental variable "XMP_ONESIDED_HEAP_SIZE"
```

The `XMP_ONESIDED_HEAP_SIZE` specifies the memory size that is allocated at the program start for a onesided function. The above error message indicates that the allocated memory size is too small. The default size is **16 Mbytes**. If you set a new value, execute as follows.

```
$ export XMP_ONESIDED_HEAP_SIZE=32M
```

XMP_ONESIDED_STRIDE_SIZE (Only GASNet)

The `` `XMP_ONESIDED_STRIDE_SIZE `` must be set when the following execution error occurs.

```
[ERROR] Memory size for coarray stride transfer is too small.
        Please set the environmental variable "XMP_COARRAY_STRIDE_SIZE"
```

The `XMP_ONESIDED_STRIDE_SIZE` specifies the memory size that is allocated at the program start for stride access via coarray (for example, `a(1:N:2) = b(1:N:2)[2]`). The above error message indicates that the allocated memory size is too small. The default size is **1 Mbyte**. If you set a new value, execute an example as follows.

```
$ export XMP_ONESIDED_STRIDE_SIZE=2M
```

When using GASNet as a onesided communication library, the program allocates the memory size of the value by adding `XMP_ONESIDED_HEAP_SIZE` to `XMP_ONESIDED_STRIDE_SIZE` at the program start.

Tips

How to compile one part of code by a native compiler

You can link object files generated by a native compiler and those by the omni compiler. However, the restrictions are as follows.

C language

For example, assume you have `a.c` and `b.c`. If you want to compile `a.c` by the native compiler `mpicc` and you want to compile `b.c` by the omni compiler `xmpcc`, you can compile them individually. Note that you need to use the omni compiler to link the object files.

```
$ mpicc a.c -c
$ xmpcc b.c -c
$ xmpcc a.o b.o
```

Fortran

While the process is basically the same as with the C language, there is an additional restriction for using a module file. The omni compiler uses an `.xmod` file, which is a particular file for using a module file. Therefore, for example, even to generate a normal `.mod` file by gfortran, the `.mod` file cannot be used in the omni compiler.

In order to transform a `.mod` file to an `.xmod` file, the omni compiler provides a `T_Module` command. The `T_Module` command supports the `.mod` file generated by gfortran-4.4, 4.7, and 4.9.

In order to build the `T_Module` command, you add the `--enable-mod2xmod` option to `./configure`. Note that the build process of `T_Module` needs mpfr and gmp.

```
$ ./configure --enable-mod2xmod
```

For example, to generate `test.xmod` from `test.mod`, you execute as follows.

```
$ T_module test.mod
```

In the case of failure of installation

Confirm PATH of MPI

Confirm the setting of `PATH`, which is set to MPI commands by using the `which` command. The following is an example case in which you installed OpenMPI by the `aptitude` command in Debian GNU/Linux 8.3.

```
% which mpicc
/usr/bin/mpicc
```

If `PATH` is not set to the MPI commands, the `which` command outputs no message.

When you installed OpenMPI by the `yum` command in CentOS 7, OpenMPI is installed in `/usr/lib64/openmpi/`. Therefore, you need to set `PATH` manually as follows.

```
$ export PATH=/usr/lib64/openmpi/bin:$PATH
```

Note in the case of using GASNet except for mpi-conduit

When using GASNet except for mpi-conduit, GASNet is restricted to GASNet communication (coarray, post/wait/lock/unlock directives) and MPI communication (communication directives except for post/wait/lock/unlock directives, for example, the bcast directive) at the same time. If you want to know more details, please refer to "MPI Interoperability" of README in GASNet in detail).

Therefore, you cannot simultaneously use the two kinds of communication. In particular, the `xmp_sync_all()` function in XcalableMP/C or the `sync all` statement in XcalableMP/Fortran is inserted after communication by GASNet. In a similar way, the `barrier` directive is inserted after communication by MPI.

- XcalableMP/C

```
/* ... use GASNet as usual ... */

xmp_sync_all(&status)

/* ... use MPI as usual ... */

#pragma xmp barrier

/* ... use GASNet as usual ... */
```

- XcalableMP/Fortran

```
/* ... use GASNet as usual ... */

sync all

/* ... use MPI as usual ... */

!$ xmp barrier

/* ... use GASNet as usual ... */
```

Test programs for the omni compiler

The omni compiler prepares test programs to confirm whether the omni compiler works properly. In order to compile and execute the test programs, you execute the following commands after installing the omni compiler and setting `PATH`.

```
$ make tests  // Compile test programs
$ make run-tests  // Execute test programs
$ make clean-tests // Delete binaries of test programs
```

The test programs are generated in the `./test` directory by the `make tests` command, and the test programs by the `make run-tests` command execute on a local node. Therefore, when you use a cross-compiler, you cannot execute the test programs by the `make run-tests` command. If you want to execute the test programs by using a cross-compiler, you need to execute them on a compute node manually.

Cooperation with profiling tools

The omni compiler has a function to cooperate with profiling tools, Scalasca (we confirmed it works with Scalasca version 1.4.3) and tlog. The function is available to measure time and so on of the following directives. At the present time, the function supports only XcalableMP/C.

- loop
- reduction
- gmove
- bcast
- reflect
- barrier
- task

Profiling using Scalasca

First, you install Scalasca, and set `PATH` to the installed Scalasca.

If you want to profile all directives that exist in a code, you add the `--profile scalasca` option to the compile command.

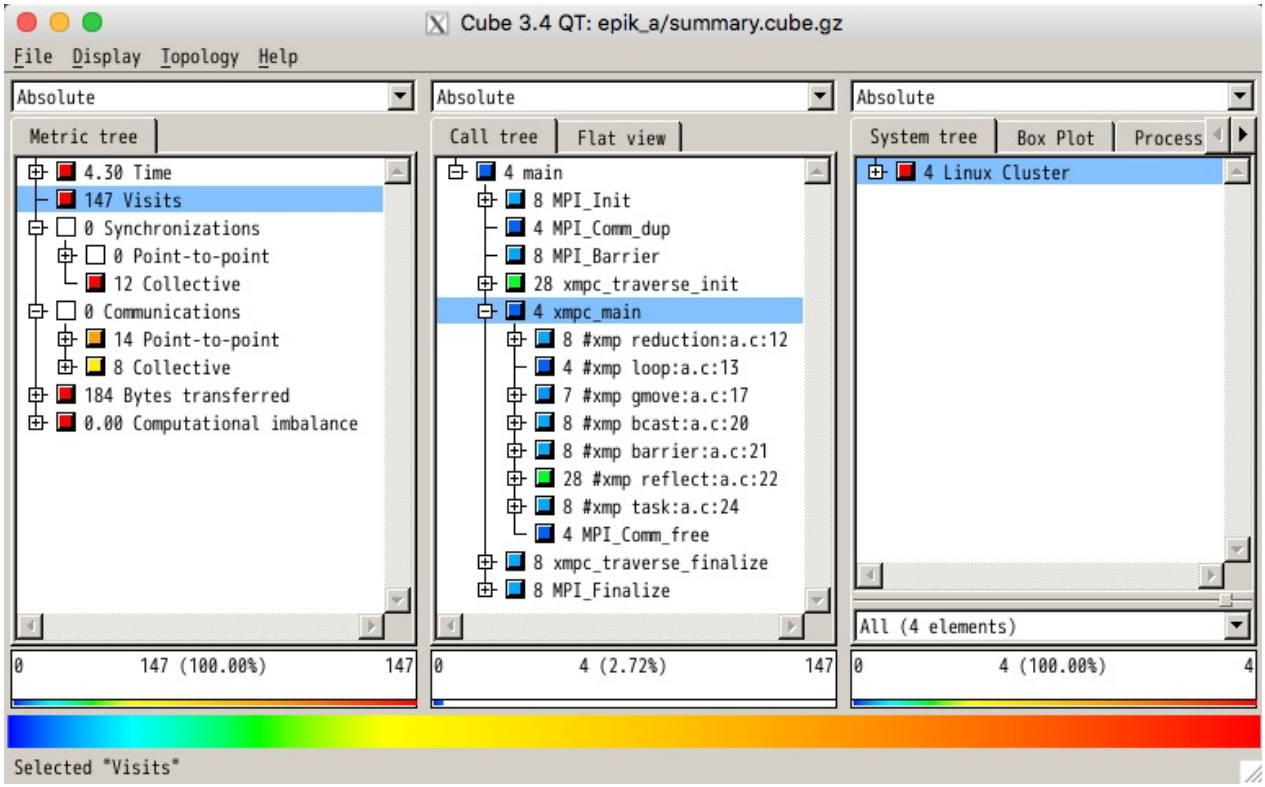
```
$ xmpcc --profile scalasca a.c
```

If you want to profile selected directives that exist in a code, you add the `profile` clause to the directive and you add the `--selective-profile scalasca` option to the compile command.

```
#pragma xmp bcast (a) profile
```

```
$ xmpcc --selective-profile scalasca a.c
```

For more information about Scalasca, please refer to the Scalasca official site.



Profiling using tlog

The tlog is automatically installed when installing the omni compiler.

If you want to profile all directives that exist in a code, you add the `--profile tlog` option to the compile command.

```
$ xmpcc --profile tlog a.c
```

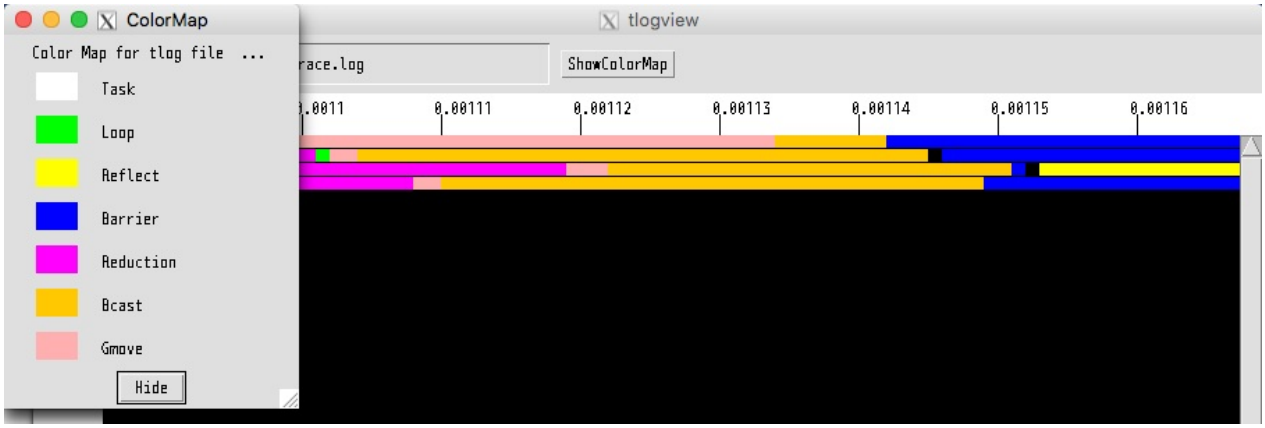
If you want to profile selected directives that exist in a code, you add the `profile` clause to the directive and you add the `--selective-profile tlog` option to the compile command.

```
#pragma xmp bcast (a) profile
```

```
$ xmpcc --selective-profile tlog a.c
```

After executing a program, `trace.log` is generated to save the profiling results. When you open the result, you use the `tlogview` command.

```
$ tlogview trace.log
```


ColorMap
Color Map for tlog file ...
Task
Loop
Reflect
Barrier
Reduction
Bcast
Gmove
Hide
tlogview
race.log
ShowColorMap
0.00111
0.00112
0.00113
0.00114
0.00115
0.00116

Restrictions

On the K computer, FX100, and FX10

- The number of coarrays in an application is 508 or less
- An application cannot be used in more than 82,944 processes
- Post tag value is between 0 and 14 (0 <= tag <= 14)

For module file of XcalableMP/Fortran

As "How to compile one part of code by a native compiler" in 4. Tips, a module file must be in an `.xmod` format that can be analyzed by the omni compiler.

Development Status