FP2C Manual 0.0.2

Miwako TSUJI

2014.03.27

Contents

Chapter 1

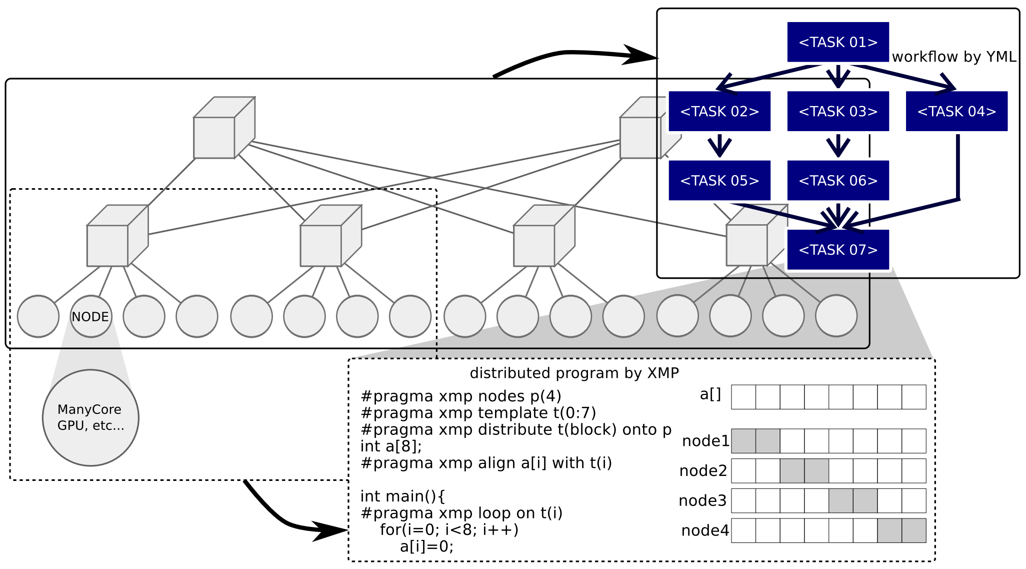
Introduction

FP2C (Framework for Post-Petascale Computing) is a development and execution environment which supports multi program methodologies across multi architectural levels. It introduces a PGAS parallel programming language called XcalableMP (XMP) to describe tasks into a workflow environment called YML.

FP2C is composed of three layers (Fig. 1.1):

1. workflow programming
2. parallel and distributed programming
3. shared-memory parallel programming/accelerator.

The tasks are expected to be executed on sub-clusters or groups of nodes which are tightly connected. These tasks would be hybrid programs with distributed and shared programming models. The workflow scheduler among the sub-clusters or groups invokes and manages the tasks.



```
#pragma xmp nodes p(4)
#pragma xmp template t(0:7)
#pragma xmp distribute t(block) onto p
int a[8];
#pragma xmp align a[i] with t(i)

int main(){
#pragma xmp loop on t(i)
    for(i=0; i<8; i++)
        a[i]=0;
```

Figure 1.1: Overview

Chapter 2

Change

2014.03.27

- XMP FORTRAN Support
- XMP-dev/StarPU Support

Chapter 3

Install

3.1 Before install

We assume that you'll install all the stuffs under the `$PREFIX` directory. For example, `PREFIX=/home/usrname/local`

3.2 Expat

3.2.1 Install Expat

Get expat-2.1.0 (or later) from `http://sourceforge.net/projects/expat/`.

```
$ tar xfz expat-2.0.1.tar.gz
$ cd expat-2.1.0
$ ./configure --prefix=$PREFIX
$ make
$ make install
```

Note: It is recommended that you install expat by yourself even if your system already has expat elsewhere.

3.3 XcalableMP (XMP)

3.3.1 Requirements for XMP

- Lex, Yacc
- C Compiler (supports C99)
- Java Compiler (latest)
- Apache Ant (1.8.1 or later)

- MPI Implementation (supports MPI-2)
- libxml2
- GASNet (If you want to use coarray functions)

3.3.2 Install XMP

Get Omni XcalableMP Compiler from `http://www.xcalablemp.org/` or `http://sourceforge.net/projects/xcalablemp/files/`.

```
$ tar xfz omnixmp-Rxxxx.tar.gz
$ cd omnixmp-Rxxxx
$ export JAVA_HOME=${your_java_dir}
$ ./configure --prefix=$PREFIX --with-libxmlDir=${your_xml_dir} --with-mpicc=${your_mpi_
$ make
$ make install
```

Note: ${your_xml_dir} is `/usr` if you have `xmlreader.h` at `/usr/include/libxml2/libxml/`.

Note: You can omit the `--with-mpicc` option if your MPI-C compiler command is `mpicc`. Otherwise, specify your command.

For more details, see `omnixmp-?.?.?/README`.

3.4 OmniRPC-MPI

3.4.1 Requirements for OmniRPC-MPI

- Flex
- C Compiler (supports C99)
- Java Compiler (latest)
- MPI Implementation (supports MPI-2)

Basically, you may already have all the requirements if you have installed XMP successfully.

3.4.2 Install OmniRPC-MPI

Get OmniRPC-MPI from `http://yml.prism.uvsq.fr/`.

```
$ tar xfz omnirpc-x.x.x.tar.gz
$ cd omnirpc-x.x.x
$ ./configure --prefix=$PREFIX --enable-gcc --with-cc=mpicc --with-opt=-fPIC
$ make
$ make install
```

Note: Please replace mpicc with your MPI-C command if it is not mpicc.

If you have the error about `YYSTYPE`, it may be a bug in your flex. Edit the last few lines in `src/omrpc-gen/y.tab.h`, `src/omrpc-gen-mpi/y.tab.h`, and `src/omrpc-gen-xmp/y.tab.h` as follows:

```
-- Before ------------------------------
#if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED

# define yystype YYSTYPE /* obsolescent; will be withdrawn */
# define YYSTYPE_IS_DECLARED 1
# define YYSTYPE_IS_TRIVIAL 1
#endif

extern YYSTYPE yylval;
----------------------------------------

-- After -------------------------------
#if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
typedef int YYSTYPE;
# define yystype YYSTYPE /* obsolescent; will be withdrawn */
# define YYSTYPE_IS_DECLARED 1
# define YYSTYPE_IS_TRIVIAL 1
#endif

extern YYSTYPE yylval;
----------------------------------------
```

3.5 Libutil

3.5.1 Requirements for libutil

- Expat
- pkg-config (Pkg-config is required to install YML with libutil)

Note: You have installed Expat in section 3.2.1.

3.5.2 Install libutil

Get libutil from `http://yml.prism.uvsq.fr/`.

```
$ tar xfz libutil-0.1.5.tar.gz
$ cd libutil-0.1.5
$ ./configure --prefix=$PREFIX \
  --with-expat-incdir=$PREFIX/include --with-expat-libdir=$PREFIX/lib
$ make
$ make install
$ export PKG_CONFIG_PATH=$PREFIX/lib/pkgconfig
```

3.6 YML

3.6.1 Requirements for YML

You may have all the requirements if you have installed the above items successfully. One note is that **shared** MPI libraries such as libmpi.**so**, are required.

3.6.2 Install YML

Get YML from `http://yml.prism.uvsq.fr/`.

```
$ tar xfz yml-2.x.x.tar.gz
$ cd yml-2.x.x
$ ./configure --prefix=$PREFIX \
  --with-expat-incdir=$PREFIX/include --with-expat-libdir=$PREFIX/lib \
  --disable-xtremweb-net
$ make
$ make install
```

If the names of your MPI libraries are not `libmpi.a/libmpi.so`, then edit the following line at `configure` file before `./configure`:

```
MPI_LIBS="-L$mpi_lib_dir -lmpi"
```

For example, if you use mvapich,

```
MPI_LIBS="-L$mpi_lib_dir -lmpich"
```

or

```
MPI_LIBS="-L$mpi_lib_dir -lmpich -lmpl"
```

If you have the error of the conflict of MPI::Init, then set `CPPFLAGS` to avoid `MPICXX`. For example, for mvapich,

```
env CPPFLAGS=-DMPICH_SKIP_MPICXX ./configure --prefix=$PREFIX \
--with-expat-incdir=$PREFIX/include --with-expat-libdir=$PREFIX/lib \
--disable-xtremweb-net
```

For OpenMPI,

```
env CPPFLAGS=-DOMPI_SKIP_MPICXX ./configure --prefix=$PREFIX \
--with-expat-incdir=$PREFIX/include --with-expat-libdir=$PREFIX/lib \
--disable-xtremweb-net
```

If you have the error about the search path to mpi.h, then add it `CPPFLAGS`,

```
env CPPFLAGS=-I/your/mpi/incdir ./configure --prefix=$PREFIX \
--with-expat-incdir=$PREFIX/include --with-expat-libdir=$PREFIX/lib \
--disable-xtremweb-net
```

Chapter 4

Configuring

4.1 Configuration for YML

The configuration file of YML are located in `$PREFIX/etc/yml`. It consists of a set of xml file with the extension .xcf. For FP2C (XMP/YML) software configuration, `yml.xcf` and `mpi.xcf` must be edited.

4.1.1 yml.xcf

You must edit the group named **Backend** as follows:

```
<?xml version="1.0" ?>
<config>
  <!-- Path information -->
  .....
  ......
  <!-- Backend module -->
  <group name="Backend" >
    <entry name="module"   value="MpiBackend" />
    <entry name="init"     value="mpi" />
  </group>
  ......
```

4.1.2 mpi.xcf

You must replace the value of `hostfileDir` with the path of your OmniRPC-MPI hostfile. Refer section 4.2 for OmniRPC hostfile.

If you don't want to output the execution log of OmniRPC-MPI, then replace "yes" with "no" for `verbose`.

```
<?xml version="1.0" ?>
<config>
  <group name="general">
```

```
    <entry name="execution-catalog"  value="DefaultExecutionCatalog" />
    <entry name="version"            value="1.0" />
    <entry name="maxNumberRequest"   value="500" />
    <entry name="hostfileDir"        value="/home/usr/.omrpc_registry" />
    <entry name="verbose"            value="yes" />
  </group>
</config>
```

4.2 Configuration for OmniRPC-MPI

Make a directory `.omrpc_registry` in the user's home directory. In `.omrpc_registry`, configuration files for OmniRPC are stored.

```
$ mkdir $HOME/.omrpc_resistry
```

Put a host file named `hosts.xml` at `.omrpc_registry`:

```
<?xml version="1.0" ?>
<OmniRpcConfig>
<Host name="127.0.0.1" arch="i386" os="linux">
<Agent invoker="mpi" />
<JobScheduler type="rr" maxjob="1000" />
</Host>
</OmniRpcConfig>
```

The agent invoker "mpi" indicates that remote programs, which execute tasks in a workflow, are invoked in parallel via MPI_Comm_spawn.

Puts a file named `nodes` at `.omrpc_registry`. The `nodes` file includes the list of nodes to be used by a workflow application.

Puts a stub file named `stubs`, which consists of the module name, function name and path to the remote executable program, at `.omrpc_registry`. For example, `stubs` would be

```
add addxy /home/usr/local/bin/remote_prog1.rex
add addyz /home/usr/local/bin/remote_prog1.rex
mul mulxy /home/usr/local/bin/remote_prog2.rex
mul mulyz /home/usr/local/bin/remote_prog2.rex
...
```

This will be also explained in section 5.1.4.

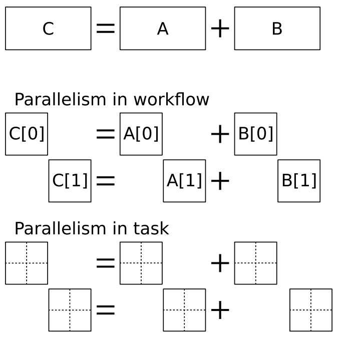
Chapter 5

Tutorial

YML is a development and execution environment for a workflow. Yml_component and yml_compiler are used in the development phase and yml_scheduler is used in the execution phase.

5.1 Development

As an example, we consider a simple application $C := A + B$, where A, B and C are matrices.



Let divide each matrix into sub-matrices $A[0], A[1], \cdots$. A task "add" calculate $C[i] = A[i] + B[i]$. In the task, each of sub-matrix is distributed and calculated in parallel.

In this example, there are two kinds of task, `init` and `add`.

5.1.1 Abstract

Edit `init.query` and `add.query` as follows:

```
<?xml version="1.0" ?>
<component type="abstract" name="init">
<params>
<param name="A" type="Matrix" mode="out" />
<param name="B" type="Matrix" mode="out" />
<param name="C" type="Matrix" mode="out" />
</params>
</component>
```

```
<?xml version="1.0" ?>
<component type="abstract" name="add">
<params>
<param name="A" type="Matrix" mode="in" />
<param name="B" type="Matrix" mode="in" />
<param name="C" type="Matrix" mode="inout" />
</params>
</component>
```

In the abstract query, the name of task and parameters for the task are defined. For the parameter of task, its name, type and mode are defined.

In this example, the type "Matrix" indicates that the corresponding variable is a real value matrix (two-dimensional array) and aligned with an xmp-template. See section for the detailed information of types supported.

To register an abstract query into DefaultDevelopmentCatalog of YML

```
$ yml_component --force init.query
$ yml_component --force add.query
```

5.1.2 Implementation

The implementation query is used to describe the computation executed remotely, i.e. task. Yml_component generates an executable file combining implementation and abstract queries.

To describe parallel procedure in tasks easily, we support XcalableMP (XMP). For the detal of XcalableMP, see `http://www.xcalablemp.org/`.

Edit `init_impl.query` and `add_impl.query` as follows:

```
<?xml version="1.0"?>
<component type="impl" name="init" abstract="init">
<impl lang="XMP" nodes="CPU:(2,2)">
<templates>
<template name="t" format="block,block" size="8,8"/>
</templates>
<distribute>
<param template="t" name="A(8,8)" align="[i][j]:(j,i)"/>
<param template="t" name="B(8,8)" align="[i][j]:(j,i)"/>
<param template="t" name="C(8,8)" align="[i][j]:(j,i)"/>
</distribute>
<header>
<![CDATA[
#include<xmp.h>
]]>
</header>
<source>
<![CDATA[
  int i,j,n;
  n=8;
#pragma xmp loop (j,i) on t(j,i)
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      C[i][j]=0.0;
      A[i][j]=1.0;
      B[i][j]=i*n+j+1;
    }
  }
]]>
</source>
</impl>
</component>
```

```
<?xml version="1.0"?>
<component type="impl" name="add" abstract="add">
<impl lang="XMP" nodes="CPU:(2,2)">
<templates>
<template name="t" format="block,block" size="8,8"/>
</templates>
<distribute>
<param template="t" name="A(8,8)" align="[i][j]:(j,i)"/>
<param template="t" name="B(8,8)" align="[i][j]:(j,i)"/>
<param template="t" name="C(8,8)" align="[i][j]:(j,i)"/>
</distribute>
<header>
<![CDATA[
#include<xmp.h>
]]>
</header>
<source>
<![CDATA[
  int i,j,n;
  n=8;
#pragma xmp loop (j,i) on t(j,i)
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      C[i][j]=A[i][j]+B[i][j];
    }
  }
]]>
</source>
</impl>
</component>
```

To register an implement query into DefaultDevelopmentCatalog of YML

```
$ yml_component --force init_impl.query
$ yml_component --force add_impl.query
```

During the registration process, yml_component generates executable files and puts them at `$PREFIX/var/yml/dr/binaries/`.

The `<templates>` and `<distribute>` elements specify distribution of multi-dimensional arrays over the nodes specified at `nodes` attribute in `<impl>` element. See the section 6.5 for the detail of data mapping.

For the options of `yml_component`, see `yml_component --help`.

To develop tasks, libaries such as BLAS, scalapack, etc... can be used. See section .

To use XMP for Fotran or XMP-dev/StarPU, see the appendex.

5.1.3 Application

To describe workflow applications, YvetteML language is supported. See section 6.4 for the brief introduction of YML, see `http://yml.prism.uvsq.fr/data/manual/1.0/The-YvetteML-I` for the detail of YvetteML.

Edit "test.query" as follows:

```
<?xml version="1.0"?>
<application name="test">
<graph>

n:=2;
par
  par(i:=0;n-1)
  do
    compute init(A[i],B[i],C[i]);
    notify(init[i]);
  enddo
//
  par(i:=0;n-1)
  do
    wait(init[i]);
    compute add(A[i],B[i],C[i]);
  enddo
endpar

</graph>
</application>
```

Then, compile it

```
$ yml_compiler test.query
```

`test.query.yapp` would be generated. It contains the description of the application in its intermediate representation adapted to its execution using the scheduler.

For the options of `yml_compiler`, see `yml_compiler --help`.

5.1.4 Binding

If a task$_i$ which uses 4 processes is followed by a task$_b$ which also requires 4 processes, then it is not necessary to finalize task$_i$ and spawn task$_j$. In such case, task$_i$ and task$_j$ are linked to a single remote program, and the processes which have finished task$_i$ would start task$_j$ on request from yml_scheduler linked with OmniRPC-MPI library. We provide a tool to bind such tasks, i.e. object files include task functions, into a single executable file.

After you finished steps written in section 5.1.1 – 5.1.3, use `omrpc-register-yml` with the name of compiled application query:

```
$ omrpc-register-yml test.query.yapp
```

Put the last few lines from the above command into `.omrpc_registry/stubs` , which was described in section 4.2.

5.2 Execution

To execute a workflow application, yml_scheduler should be executed by mpirun (or mpiexec, orterun, etc...).

An example of Open MPI is

```
$ mpirun -n 1 -machinefile <filename> yml\_scheduler test.query.yapp
```

For the detail of MPI-2 dynamic process management, refer the user guide of your MPI implementation.

After the execution of the workflow application, you can find directory `run<num>`, where `<num>` is a serial number. In `run<num>`,

Chapter 6

Appendix

6.1 Supported languages

- C++ : `lang="C++"`
- XMP for C : `lang="XMP"`
- XMP for Fortran : `lang="XMF"`
- XMP-dev/StarPU for C: `lang="XDS"`

6.2 Types in abstract query

In this section, types supported by YML (YML with C++) and FP2C (YML with XMP) is shown. These types are specified in abstract query (See section 5.1.1). You can also define your original types.

6.2.1 types for C++

real	double-precision floating-point
integer	integer
string	array of character

6.2.2 types for XMP & XMP-dev/StarPU for C

real	double-precision floating-point
integer	integer
char	character
Matrix	two dimensional array, real, aligned with template
Vector	one dimensional array, real, aligned with template

6.2.3 types for XMP for Fortran

real8	double-precision floating-point
integer	integer
real8vector	one dimensional array, real, aligned with template
real8matrix	two dimensional array, real, aligned with template

6.3 How to use libraries

To develop tasks, libraries such as BLAS, scalapack, etc... can be used.

Firstly, a text file with a `.txt` extension is edited and putted under `$PREFIX/var/yml/DefaultExecutionCatalog/generators/XMP/lib/`. The `.txt` file consists of two lines, (1) the list of directories to be searched for header files with `-I` option, (2) the list of libraries. For example,

```
-I/usr/local/include
/usr/local/lib/libscalapack.a /usr/local/lib/liblapack.a /usr/local/lib/blas_LINUX.a
```

Then give the name of the file to `libs` attribute in implementation query. For example, if you have `mylibrary.txt`,

```
<?xml version="1.0"?>
<component type="impl" name="test" abstract="test">
<impl lang="XMP" nodes="CPU:(4)" libs="mylibrary">
<templates>
...
```

One or more txt files can be used for different libraries.

```
<impl lang="XMP" nodes="CPU:(4)" libs="mymath mylib">
```

6.4 YvetteML

This section briefly review about YvetteML Graph Description Language. For more details, see the YML manual `http://yml.prism.uvsq.fr`.

compute (call task)

To execute a task on remote nodes, call the name of task declared in the abstract.query with `compute` keyword, and give input/output parameters.

```
  compute add(a,b,result);
```

par (parallel section)

Parallel sections are declared with `par endpar` keywords and are separated by a // separator.

```
par
  compute add(1,2,result[1]);
//
  compute add(3,4,result[2]);
//
  compute add(5,6,result[3]);
endpar
```

Parallel-loop is also supported by `par-do enddo` keywords.

```
par (i:=1; 10)
  do
  compute test(i,result[i]);
enddo
```

ser (sequential loop)

Sequential-loop can be described with `seq` keyword.

```
seq (i:=1; 10)
  do
  compute test(i,result[i]);
enddo
```

if then, else endif (conditional)

Definition of conditional branches based on a boolean condition.

```
if i neq j then
  compute test1(i,j,result[k]);
else
  compute test2(i,j,result[k]);
endif
```

notify, wait (events manipulation)

The events can be explicitly managed using two constructions called `notify` and `wait`.

```
par
  par (i:=1; 10)
    do
    compute init(A[i]);
    notify(init[i]);
  enddo
//
  par (i:=1; 10)
    do
    wait(init[i]);
    compute test(A[i],result[i]);
  enddo
endpar
```

6.5 Data Mapping in a task

The declarations of nodes, template and distribution elements and attributes in implementation query are interpreted as the nodes, template, distribution and align directives in XMP. For example,

```
<?xml version="1.0"?>
<component type="impl" name="init" abstract="init">
<impl lang="XMP" nodes="CPU:(2,2)">
<templates>
<template name="t" format="block,block" size="8,8"/>
</templates>
<distribute>
<param template="t" name="A(8,8)" align="[i][j]:(j,i)"/>
</distribute>
```

is translated into

```
#pragma xmp nodes _XMP_default_nodes(2,2)

#pragma xmp template t(0:7,0:7)
#pragma xmp distribute t(block,block) onto _XMP_default_nodes

XMP_Matrix A[8][8];
#pragma xmp align A[i][j] with t(j,i)
```

The data mapping information is used to export and import data to each task. Programmers do not have to program data export/import by themselves. Note that for this automatic data distribution, the distribution format `cyclic` has not been suppoted yet.

If you want to mix different topologies of nodes in a single task, then you can specify `nodes` attribute in the `template` element in addition to the default node topology.

```
<?xml version="1.0"?>
<component type="impl" name="init" abstract="init">
<impl lang="XMP" nodes="CPU:(2,2)">
<templates>
```

```
<template name="t" format="block,block" size="8,8"/>
<template name="u" format="block"       size="8"   nodes="p(4)"/>
</templates>
<distribute>
<param template="t" name="A(8,8)" align="[i][j]:(j,i)"/>
<param template="u" name="B(8)"   align="[i]:(i)"/>
</distribute>
```

Note that the order for the template in XMP indices is based on Fortran conventions.

For the variables which are used only in a task, i.e. for the variables which are not input/output parameters defined in an abstract query, their distributions can be specified in the `<header>` element.

```
<?xml version="1.0"?>
<component type="impl" name="init" abstract="init">
<impl lang="XMP" nodes="CPU:(2,2)">
<templates>
<template name="t" format="block,block" size="8,8"/>
</templates>
<distribute>
<param template="t" name="A(8,8)" align="[i][j]:(j,i)"/>
</distribute>
<header>
<![CDATA[
double tmp[8][8];
#pragma xmp align tmp[i][j] with t(j,i)
]]>
</header>
<source>
<![CDATA[
  int i,j;
#pragma xmp loop (i,j) on t(j,i)
  for(i=0;i<8;i++){
    for(j=0;j<8;j++){
      tmp[i][j]=A[i][j];
    }
  }
  ......
  ......
]]>
</source>
</impl>
</component>
```

6.6 Tips to use XMP for Fortran

```
<?xml version="1.0" encoding="UTF-8"?>
<component type="abstract" name="xmpf_out">
<params>
<param name="v1" type="real8vector" mode="in" />
</params>
</component>
```

```
<?xml version="1.0"?>
<component type="impl" name="add" abstract="add">
<impl lang="XMF" nodes="CPU:(2)">
  <templates>
    <template name="t" format="block" size="10"/>
  </templates>
  <distribute>
    <param template="t" name="v1(10)" align="(i):(i)"/>
  </distribute>
<header>
         integer i
</header>
  <source>
<![CDATA[
!$xmp loop on t(i)
         do i=1,10
           print *,"v1(i)=",v1(i)
         end do
]]>
  </source>
</impl>
</component>
```

These are abstract and implementation query codes written with XMP Fortran. There are some remarks:

- The variables used within a task must be written between `<header>` ∼ `</header>`.
- The template declarations must be based on the "XMP-Fortran" specification, not the "XMP-C" specification, i.e, it must **not** be "`[i][j]:(j,i)`" but "`(j,i):(j,i)`".

6.7 Tips to use XMP-dev/StarPU for C

```
<?xml version="1.0"?>
<component type="impl" name="dgemm_prod_mat" abstract="dgemm_prod_mat">
<impl lang="XDS" libs="starpu" nodes="CPU:(1,1)">
<templates>
<template name="t" format="block,block" size="256,256"/>
</templates>
<distribute>
<param template="t" name="A(256,256)" align="[i][j]:(j,i)"/>
<param template="t" name="B(256,256)" align="[i][j]:(j,i)"/>
<param template="t" name="C(256,256)" align="[i][j]:(j,i)"/>
</distribute>
<header>
<![CDATA[
#include<xmp.h>
#define N 256
static double B1[256][256];
#pragma xmp align B1[i][j] with t(j,i)
#pragma xmp shadow B1[*][0:0]
]]>
</header>
<source>
<![CDATA[
  int    i,j,k,n;
  // ....
  // Your soruce code written in XMP-dev/StarPU
]]>
</source>
<footer />
</impl>
</component>
```

This is an example of the implementation query code for XMP-dev/StarPU. You have to give the library file `starpu.txt` under your catalog directory, generally `$PREFIX/var/yml/DefaultExecutionCatalog/generators/XDS/lib`. The `starpu.txt` should be consist of two lines. At the first line of your `starpu.txt`, you have to specify your include-directory for CUDA and StarPU. At the second line, you have to specify the full pathes of your XMP and XMP-dev libraries, StarPU libraries CUDA libraries. For example,

```
-I$XDS_PREFIX/include -I$XDS_PREFIX/include/starpu/1.1
$XDS_PREFIX/lib/libxmp.a $XDS_PREFIX/lib/libxmp_gpu.a  \\
-L$XDS_PREFIX/lib -L/opt/CUDA/4.2.9/cuda/lib64 \\
 $XDS_PREFIX/lib/libstarpu-1.1.a -lcudart -lcublas \\
-lcuda -lhwloc -lstdc++
```

Another important remark is that the current XMP-compiler implementation avairable at `www.xcalablemp.org/` do bf not include XMP-dev/StarPU. So, you can ask the compiler implementation to the XMP-dev/StarPU development team, `odajima at hpcs.cs.tsukuba.ac.jp`.