How to install the UpScale SDK compilation framework for the Kalray MPPA Workstation

On a Linux machine

v1.0, January 2017

SCORDINO, Claudio
ROYUELA, Sara
QUIÑONES, Eduardo

Table of contents

Requirements

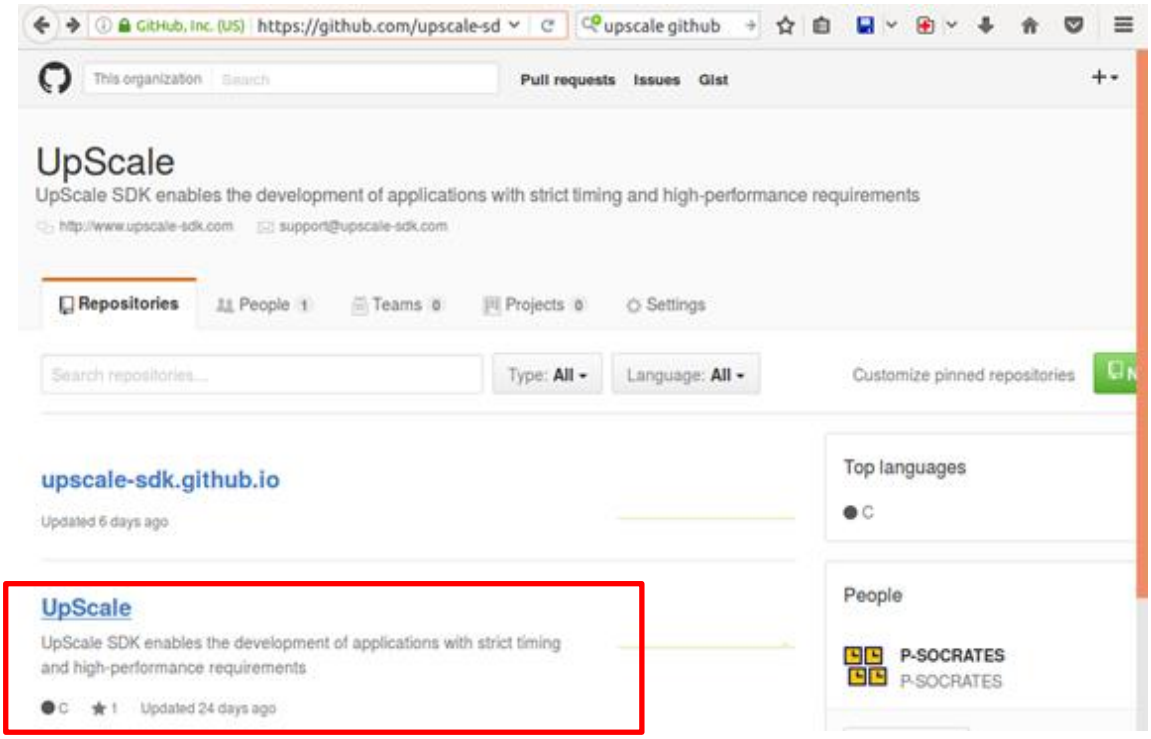
This manual uses (as example) an Ubuntu 16.04.1 distribution. For other distributions you will need to access the corresponding repositories for the required software.

Your Linux machine will need the following software installed:

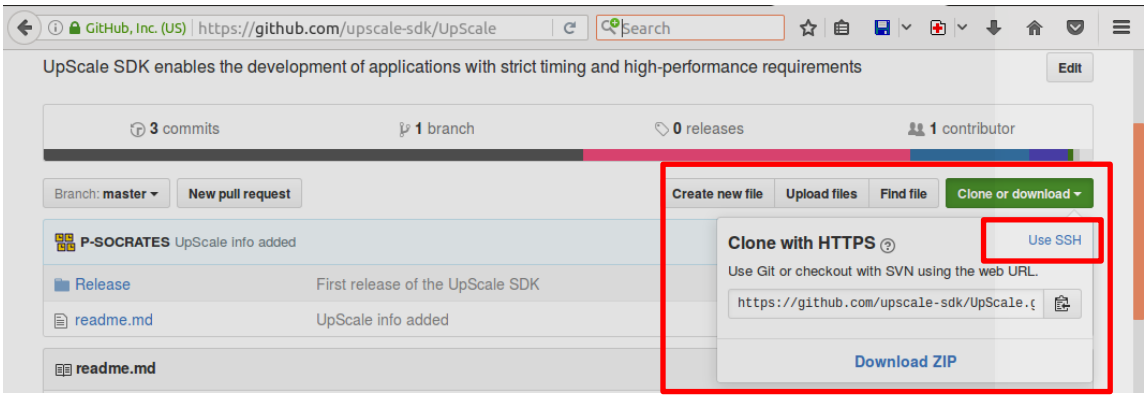
- **Git**: `sudo apt install git`
- **Autoconf**: `sudo apt install autoconf`
- **Libtool**: `sudo apt install libtool`
- **Sqlite**: `sudo apt install libsqlite3-dev`
- **Flex** *(optional)*: `sudo apt install flex`
- **Bison** *(optional)*: `sudo apt install bison`
- **Gperf** *(optional)*: `sudo apt install gperf`
- K1 tools:
    - **k1-elf-gcc** version 4.9.3 20141104 or higher
    - **k1-rtems-gcc**
    - **k1-rtems-objdump**
    - **k1-rtems-objcopy**
    - **k1-gcc**
    - **k1-create-multibinary**
    - **k1-jtag-runner**

Github repository

Go to the UpScale website on https://github.com/upscale-sdk/ and access the UpScale repository by clicking the UpScale link.

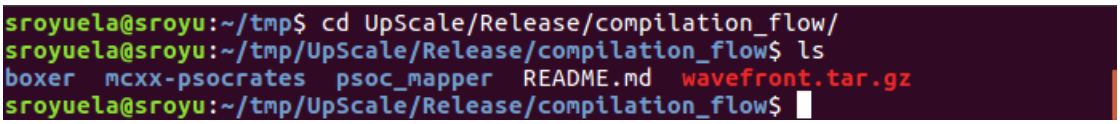


Once there, you can get the Git URL by clicking the green button on your right side that says "`Clone or download`". There are two protocol options: HTTPS and SSH (Use the "`Use SSH`"/"`Use HTTPS`" link to swap between these options). We recommend using HTTPS, so firewalls won't affect you.



Open a terminal and navigate to the location where you want to install the repository.
Once there, type "`git clone`" followed by the link you just copied (you will need *git* installed). Type enter and wait until the command finishes. A new directory called "`UpScale`" will be created.

The compilation framework is within the directory "`UpScale/Release/compilation_flow`". Navigate to that location.



ERIKA Enterprise

ERIKA Enteprise is the minimal whilst efficient Real-Time Operating System (RTOS) used by the PSOCRATES SDK to schedule the tasks execution on the computing clusters of the many-core platform. Before installing Mercurium for this project, you will need *Erika-enterprise* operating system to be installed. The installation process for the RTOS will also take care of installing the lightweight OpenMP runtime.

To build the SDK libraries containing both the ERIKA Enterprise RTOS and the OpenMP runtime, enter the directory "`UpScale/Release/execution_stack/erika-enterprise-rtems`" and type **`make`**. This command will create the "`UpScale/Release/execution_stack/erika-enterprise-rtems/psoctools`" directory containing:

- A subdirectory `include/` containing the ERIKA Enterprise headers
- A subdirectory `lib/` containing the SDK runtime libraries **`libee.a`**, **`libpsocomp.a`** and **`libpsocoffload.a`**

The path of the directory "`UpScale/Release/execution_stack/erika-enterprise-rtems/psoctools`" must be given to Mercurium during the setup (see next paragraphs).

Note 1. The compilation of the RTOS and the off-load mechanism requires the k1-elf-gcc compiler to be at version *4.9.3 20141104* or higher.

Note 2. The default installation process described in this section relies on the RTEMS operating system on the I/O cores of the MPPA platform. There is also a (still experimental) support for the Linux OS on the I/O cores. To build ERIKA Enterprise and the OpenMP runtime when the I/O cores execute the Linux OS, overwrite the directories "`UpScale/Release/execution_stack/erika-enterprise-rtems/libgomp/`" and "`UpScale/Release/execution_stack/erika-enterprise-rtems/libpsocoffload/`" with the equivalent directories available in the directory "`UpScale/Release/execution_stack/erika-enterprise-linux-experimental/`". Then, build the support as explained above.

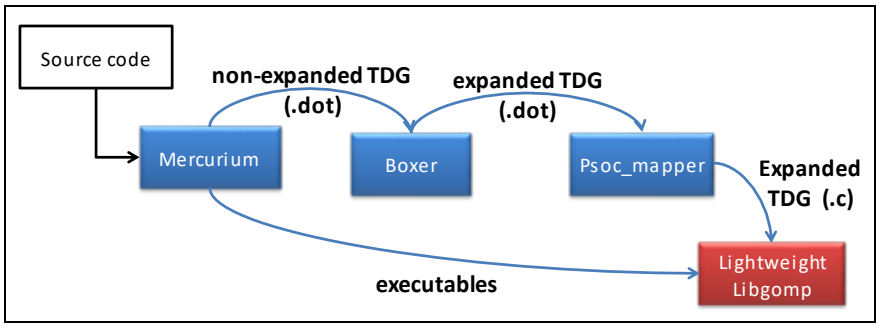
Compilation framework

The compilation framework is composed of three tools: *Mercurium*, *Boxer* and *Psoc_mapper*.

Mercurium is a source-to-source compiler that transforms the source code in such a way that specific back-end compilers can later be used to generate the executables to be run in the IO as well as the executables to be run in the Cluster. The compiler also generates a non-expanded version of the Task Dependency Graph (TDG) in DOT format that will be consumed by Boxer.

Boxer is the tool that expands the TDG generated by Mercurium into a complete TDG, also in DOT format. This TDG will be consumed by *Psoc_mapper*.

Psoc_mapper is the tool that, given a TDG in DOT format, generates the data structure containing the TDG that will be consumed by the runtime to schedule tasks.



Mercurium

The Mercurium compiler is in the directory "`mcxx-psocrates`" (under "`UpScale/Release/compilation_flow`"). We strongly recommend compiling and installing the compiler in folders other than the sources one. So, create two new folders "`mcxx-bld`", for the compilation, and "`mcxx-ins`", for the installation (you can use the "`mkdir`" command).

```
sroyuela@sroyu:~/tmp/UpScale/Release/compilation_flow$ mkdir mcxx-bld
sroyuela@sroyu:~/tmp/UpScale/Release/compilation_flow$ mkdir mcxx-ins
sroyuela@sroyu:~/tmp/UpScale/Release/compilation_flow$ ls
boxer  mcxx-bld  mcxx-ins  mcxx-psocrates  psoc_mapper  README.md  wavefront.tar.gz
sroyuela@sroyu:~/tmp/UpScale/Release/compilation_flow$
```

Enter the Mercurium sources directory and type "`autoreconf -vfi`" to generate the configure files (you will need *autoconf* and *libtool* installed).

```
sroyuela@sroyu:~/tmp/UpScale/Release/compilation_flow/mcxx-psocrates$ autoreconf -vfi
autoreconf: Entering directory `.'
autoreconf: configure.ac: not using Gettext
autoreconf: running: aclocal --force -I m4
autoreconf: configure.ac: tracing
autoreconf: running: libtoolize --copy --force
libtoolize: putting auxiliary files in '.'.
libtoolize: copying file './ltmain.sh'
libtoolize: putting macros in AC_CONFIG_MACRO_DIRS, 'm4'.
libtoolize: copying file 'm4/libtool.m4'
libtoolize: copying file 'm4/ltoptions.m4'
libtoolize: copying file 'm4/ltsugar.m4'
libtoolize: copying file 'm4/ltversion.m4'
libtoolize: copying file 'm4/lt~obsolete.m4'
autoreconf: running: /usr/bin/autoconf --force
autoreconf: running: /usr/bin/autoheader --force
autoreconf: running: automake --add-missing --copy --force-missing
configure.ac:40: installing './compile'
configure.ac:7: installing './config.guess'
configure.ac:7: installing './config.sub'
configure.ac:18: installing './install-sh'
configure.ac:18: installing './missing'
Makefile.am: installing './depcomp'
doc/Makefile.am:27: installing 'doc/texinfo.tex'
autoreconf: Leaving directory `.'
sroyuela@sroyu:~/tmp/UpScale/Release/compilation_flow/mcxx-psocrates$
```

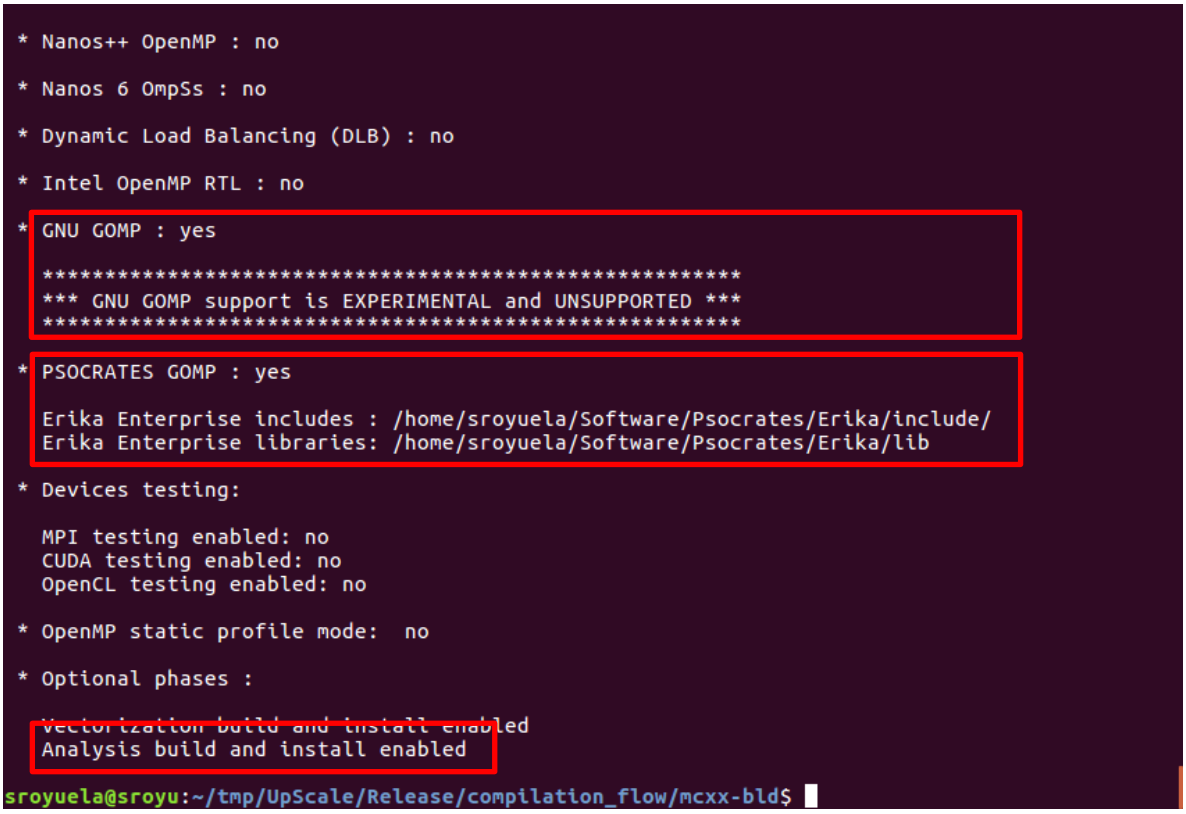
After that, go to the build directory you created before (`mcxx-bld`) and configure the project. The required flags are listed below:

- `--prefix=<<installation-path>>`
- `--enable-tl-openmp-gomp`
- `--with-erika-enterprise=<<path-to-erika-headers-and-libs>>`

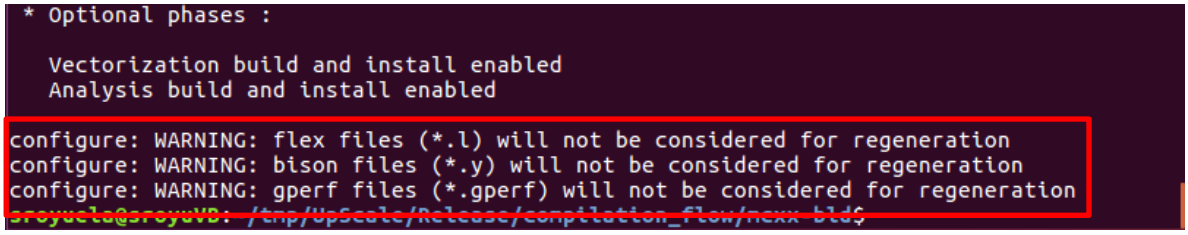
The full command will be "`../mcxx-psocrates/configure -- prefix=<<installation-path>> --enable-tl-openmp-gomp --with-erika-enterprise=$UPSCALE_HOME/UpScale/Release/execution_stack/erika-enterprise-rtems/psoctools`

```
sroyuela@sroyu:~/tmp/UpScale/Release/compilation_flow/mcxx-bld$ ../mcxx-psocrates/configure -
-prefix=/home/sroyuela/tmp/UpScale/Release/compilation_flow/mcxx-ins --enable-tl-openmp-gomp
--with-erika-enterprise=/home/sroyuela/Software/Psocrates/Erika
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking target system type... x86_64-pc-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking whether make supports nested variables... (cached) yes
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking whether gcc understands -c and -o together... yes
```

Note that at the end of the configuration *GNU GOMP*, *PSOCRATES GOMP* and *Analysis* must be enabled.



If *Flex*, *Bison* and/or *Gperf* are not installed, a warning will appear at the end of the configuration. If you are not going to modify Mercurium, don't worry about this. Otherwise, install the missing software and configure again.



Finally, you can compile will "`make`" (you will need *sqlite* installed) and install with "`make install`".

```
sroyuela@sroyu:~/tmp/UpScale/Release/compilation_flow/mcxx-bld$ make & make install
[1] 30577
  CCBUILD  lib/perish.o
  CCBUILD  lib/perish.o
  CCBUILD  lib/tpp.o
  CCBUILD  lib/tpp.o
  CCBUILD  lib/tpp
  CCBUILD  lib/tpp
  PYTHON src/frontend/cxx-asttype-nodecl.def
  TPP    src/frontend/cxx-asttype.def
tpp - a tiny preprocessor for mcxx 2.0.0
  GEN      src/frontend/cxx-asttype.h
  GEN      src/frontend/cxx-asttype.c
  TPP    src/frontend/c99.l
tpp - a tiny preprocessor for mcxx 2.0.0
  TPP    src/frontend/cxx03.l
tpp - a tiny preprocessor for mcxx 2.0.0
  TPP    src/frontend/c99.y
tpp - a tiny preprocessor for mcxx 2.0.0
  TPP    src/frontend/cxx03.y
```

Boxer

Boxer is in the directory "`boxer`" (under "`UpScale/Release/compilation_flow`"). We strongly recommend installing it in a folder other than the sources one. So, create a new folder "`boxer-ins`".

Enter the sources directory and compile and install the software with "`make PREFIX=<<installation-path>> install`".

```
sroyuela@sroyuVB:~/tmp/UpScale/Release/compilation_flow$ mkdir boxer-ins
sroyuela@sroyuVB:~/tmp/UpScale/Release/compilation_flow$ cd boxer
sroyuela@sroyuVB:~/tmp/UpScale/Release/compilation_flow/boxer$ make PREFIX=/home/sroyuela/
tmp/UpScale/Release/compilation_flow/boxer-ins/ install
cc -ggdb3 -O0   -c -o json.o json.c
cc -ggdb3 -O0   -c -o wrappers.o wrappers.c
cc -ggdb3 -O0   -c -o main.o main.c
cc -ggdb3 -O0   -c -o tree.o tree.c
cc -ggdb3 -O0   -c -o box.o box.c
cc -ggdb3 -O0   -c -o task.o task.c
cc -ggdb3 -O0   -c -o expr.o expr.c
expr.c: In function 'eval':
expr.c:273:16: warning: too many arguments for format [-Wformat-extra-args]
  fprintf(ccin, "void f(void)\n{\n", ++nfun);
                ^
cc -ggdb3 -O0  -o boxer json.o wrappers.o main.o tree.o box.o task.o expr.o
mkdir -p /home/sroyuela/tmp/UpScale/Release/compilation_flow/boxer-ins//bin #/home/sroyuel
a/tmp/UpScale/Release/compilation_flow/boxer-ins//man/man1
cp boxer /home/sroyuela/tmp/UpScale/Release/compilation_flow/boxer-ins//bin
cd cc; make install
```

PSOC_mapper

PSOC_mapper is in the directory "`psoc_mapper`" (under "`UpScale/Release/compilation_flow`"). We strongly recommend installing it in a folder other than the sources one. So, create a new folder "`psoc_mapper-ins`".

Enter the sources directory and compile and install the software with "`make PREFIX=<<installation-path>> install`".

```
sroyuela@sroyu:~/tmp/UpScale/Release/compilation_flow$ mkdir psoc_mapper-ins
sroyuela@sroyu:~/tmp/UpScale/Release/compilation_flow$ cd psoc_mapper
sroyuela@sroyu:~/tmp/UpScale/Release/compilation_flow/psoc_mapper$ make PREFIX=/home/sroyuela/t
mp/UpScale/Release/compilation_flow/psoc_mapper-ins/ install
cc sched.c psoc_mapper.c -O0 -g3 -Wall -o psoc_mapper -lm
mkdir -p /home/sroyuela/tmp/UpScale/Release/compilation_flow/psoc_mapper-ins//bin
cp psoc_mapper /home/sroyuela/tmp/UpScale/Release/compilation_flow/psoc_mapper-ins//bin
```

Environment

Remember to add all <<installation-path>> (Mercurium, Boxer and Psoc_mapper) to the PATH environment variable. You can do that by using the command "`export PATH=<<installation-path>>:$PATH`".

```
sroyuela@sroyu:~/tmp/UpScale/Release/compilation_flow$ export PATH=/home/sroyuela/tmp/UpScale/Release/compilation_flow/mc
home/sroyuela/tmp/UpScale/Release/compilation_flow/boxer-ins/bin:/home/sroyuela/tmp/UpScale/Release/compilation_flow/psoc
bin:$PATH
```

Compile and run your application

Consider the following code snippet performing a wavefront computation.

```
#include "square.h"
unsigned long int square[N][N][BS][BS];

#pragma omp declare target
long wavefront(unsigned long int square[N][N][BS][BS]) {
    #pragma omp parallel
    #pragma omp single
    {
        int i=0;
        int j=0;
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++) {
                if (j == 0 && i == 0) {
                    #pragma omp task firstprivate(i,j) \
                                     depend(inout:square[i][j])
                        sequential(i,j,square);
                } else if (i == 0) {
                    #pragma omp task firstprivate(i,j) \
                                     depend(in:square[i][j-1]) \
                                     depend(inout:square[i][j])
                        sequential(i,j,square);
                } else if (j == 0) {
                    #pragma omp task firstprivate(i,j) \
                                     depend(in:square[i-1][j]) \
                                     depend(inout:square[i][j])
                        sequential(i,j,square);
                } else {
                    #pragma omp task firstprivate(i,j) \
                                     depend(in:square[i-1][j]) \
                                     depend(in:square[i][j-1]) \
                                     depend(in:square[i-1][j-1]) \
                                     depend(inout:square[i][j])
                        sequential(i,j,square);
                }
            }
    }
}
#pragma omp end declare target

int main(void) {
    int i, n=N, bs=BS;
    init_matrix();

    GOMP_init(0);
    #pragma omp target map(tofrom:square[0:n][0:n][0:bs][0:bs]) device(0)
        wavefront(square);
    GOMP_deinit(0);

    return 0;
}
```

Find in orange the required OpenMP directives, where:

- **Parallel** directive contains the code that will execute in parallel.
- **Single** directive indicates that the code within will be executed only by one thread.
- **Task** directives contain concurrent code within the parallel region.
- **Target** directive contains the code that will be offloaded to the computing clusters.
- **Declare target** directive contains the code that is to be compiled for the cluster.
    - **Map** clause indicates the code to be copied to/from the device.
    - **Device** clause indicates the identifier of the device where the code will execute.

Find in blue the required GOMP (OpenMP RTL) calls, where:

- **GOMP_init**: creates and initializes all data structures necessary to execute OpenMP in a specific device.
- **GOMP_deinit**: frees all OpenMP data structures of a specific device.

For a complete list of the OpenMP features supported in the UpScale SDK, please review document "OpenMP supported features in UpScale SDK".

A (incomplete) Makefile for the code above will look as follows:

```
## Compiler
CC              = psocratescc
CC_CLUSTER      = psocratescc-cluster
BOXER           = boxer
MAPPER          = psoc_mapper

## Application
APP             = square                # input
TDG             = tdg_${APP}            # output

## IO part
IO_SRC          = ${CC}_${APP}.c
IO_ELF          = io_elf

## Cluster part
CLUSTER_SRC     = ${CC}_cluster_${APP}
CLUSTER_ELF     = erika

## Multibinary
MULTIBIN        = ${IO_ELF}.mpk

## Compilation process
all:
## 1. Separate IO and Cluster code
    ${CC} --tdg --debug-flags=tdg_to_json --target-mppa -c ${APP}
## 2. Extract TDG from the Cluster code
    ${BOXER} -to  ${TDG} json/*.json
    ${MAPPER} -o ${TDG}.c {TDG}_0.dot
## 3. Compile other Cluster code
    ${CC_CLUSTER} --target-mppa {TDG}.c -o {TDG}.o
## 4. Back-end compilation linking and multibinary generation
    ${CC} --v --target-mppa --sublink-output=${CLUSTER_ELF} -o ${IO_ELF}.o \
          --Wx:psocratescc-cluster:l,${TDG}.o

## Run
run:
    k1-jtag-runner --multibinary ${MULTIBIN} --exec-multibin=IODDR0:${IO_ELF}
```

The compilation process is split as follows:

1. The Mercurium compiler parses the input code and splits the code that is to be compiled for the IO from the code that is to be compiled for the Cluster (enclosed within the `declare target` directives). *psocratescc* is the Mercurium profile that performs this task (step 1 in the Makefile).
   Specifically, the following files are created during this step:
   - *cluster_square.c.c*: cluster code extracted from the `declare target` directive. The OpenMP code within has not yet been transformed.
   - *cluster_square_psocrates.report*: analysis report containing information about unsupported features found during the compilation.
   - *psocrates_cluster_square.c.c*: cluster code already lowered (this file is created from cluster_square.c.c).
   - *psocrates_square.c*: io code. The target directives have already been transformed.
   - *json/tdgs.json*: task dependency graph (not yet expanded) in JSON format.
   - *dot/**: graphs in DOT format needed for analysis purposes.

2. Other files containing only code that is to be compiled for the IO, although it does not contain code for the Cluster, shall also be compiled with the *psocratescc* profile (no such code in the example).
3. Other files containing only code that is to be compiled for the, shall be compiled with the *psocratescc-cluster* profile (step 3 in the Makefile).
4. All cluster code containing OpenMP tasks must generate its own TDG. In this case, there is only one such function, *wavefront*, so only one TDG will be generated. *Boxer* is the tool that performs this task, reading the json/tdgs.json file generated during the *psocratescc/psocratescc-cluster* compilation, and creating as many dot files as TDGs encounters (step 2 in the Makefile).
   During this step, file *tdg_square_0.dot* is created.
5. All TDGs generated by *boxer* are used together to create the file with the data structure that will be read by the runtime. *psoc_mapper* is the tool that performs this task (step 3 in the Makefile). This file must also be compiled for the Cluster (step 2 in the Makefile).
   During this step, file *tdg_square.c* is created.
6. Finally, the back-end compilation and linking, and the generation of the multibinary executable is achieved using the *psocratescc* profile together with the all the binaries generated previously and the corresponding flags (step 4 in the Makefile).
   During this step, files *erika*, *io_elf* and *io_elf.mpk* are created.

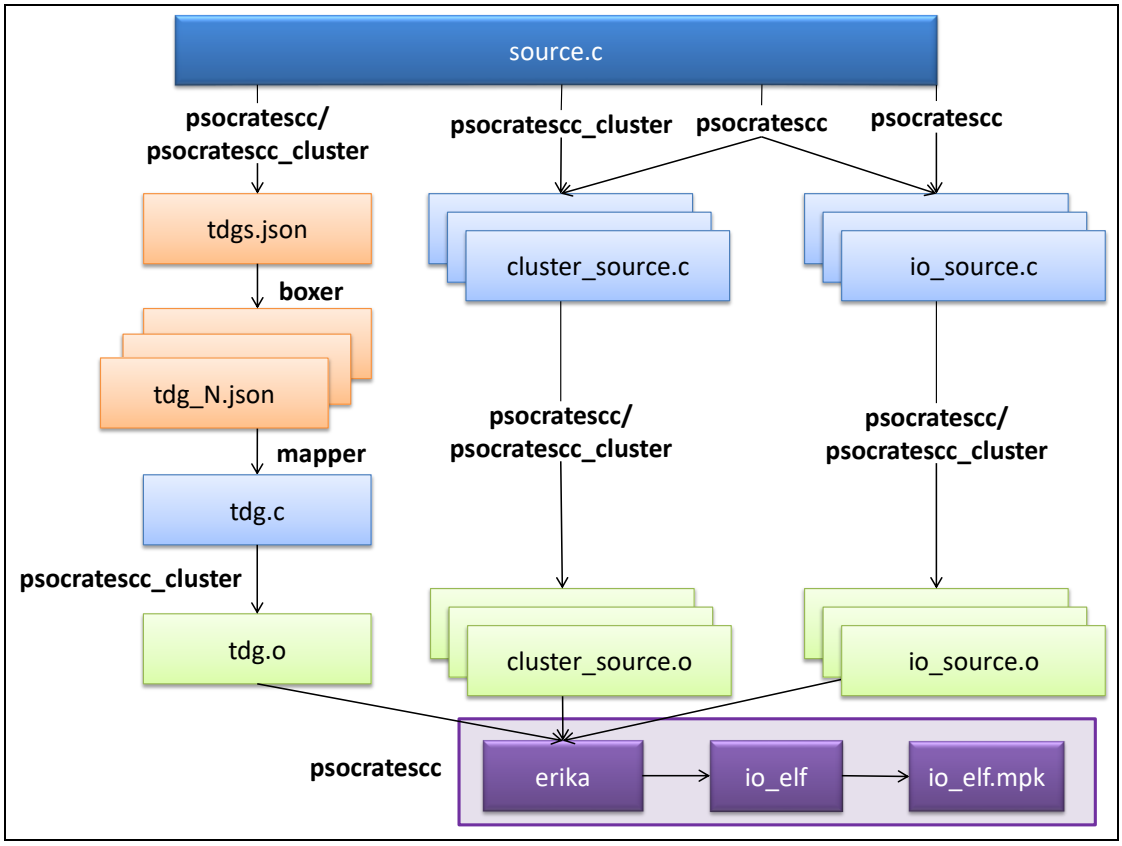


Figure 1 Compilation workflow

```
[bsc@mppa-dev wavefront]$ make
rm -f psocratescc_cluster_square.c.c psocratescc_square.c cluster_square.c.c
rm -fr json dot tdg_square.c tdg_square*.dot cluster_square_psocrates.report
rm -f psocratescc_* psocratescc-cluster_*
rm -f *~ *.o  io_elf erika io_elf.mpk io.* io_elf.* *.elf io_elf.mpk
psocratescc --tdg --debug-flags=tdg_to_json --target-mppa -c -K square.c
square.c:29:1: info: target declaration of function 'wavefront'
MPPA-I/O GOMP phase
Emitting MPPA Cluster code
GOMP phase
boxer -to tdg_square json/*.json
psoc_mapper -o tdg_square.c tdg_square_0.dot
psoc_mapper Tool version 1.0 (c) The P-SOCRATES Consortium
Using MIET by default as timing attribute
No schedulability analysis neither mapping were performed. Use -s|-d switch to enable them
psocratescc --target-mppa sequential.c -c -K -o sequential_io.o
MPPA-I/O GOMP phase
psocratescc-cluster --target-mppa sequential.c -c -K -o sequential_cluter.o
GOMP phase
psocratescc-cluster --target-mppa tdg_square.c -c -K -o tdg_square.o
GOMP phase
psocratescc --v --target-mppa --sublink-output=erika  -o io_elf square.o \
                              --Wx:psocratescc:l,sequential_io.o \
                              --Wx:psocratescc-cluster:l,tdg_square.o \
                              --Wx:psocratescc-cluster:l,sequential_cluter.o
Loading compiler phases for profile 'psocratescc'
Compiler phases for profile 'psocratescc' loaded in 0.02 seconds
k1-rtems-objdump -w -h square.o 1> /tmp/psocratescc_DcZLO1
k1-rtems-objcopy -Obinary --only-section=.mercurium square.o /tmp/psocratescc_yQlavM/multifile.tar
tar xf /tmp/psocratescc_yQlavM/multifile.tar -C /tmp/psocratescc_yQlavM .
k1-gcc -o erika -mcluster=node -mboard=developer -mhypervisor -march=k1b -mos=bare tdg_square.o sequential_cluter.o
/tmp/psocratescc_yQlavM/tag.1.psocratescc-cluster/cluster_square.c.o -L/usr/local/psoctools/lib -lpsocomp -lee -lvbs
p -lmppaipc -Xlinker --enable-new-dtags -Lgomp-dir-not-specified -Xlinker -rpath -Xlinker gomp-dir-not-specified
k1-gcc -o io_elf -mcluster=ioddr -mboard=developer -mhypervisor -march=k1b -mos=rtems square.o sequential_io.o -L/us
r/local/psoctools/lib -lpsocoffload -lmppapower -lmppanoc -lmpparouting -lpcie_queue -lmppaipc
Link performed in 0.29 seconds
k1-create-multibinary --cluster erika --boot=io_elf -T io_elf.mpk
MPPA multibinary created in 'io_elf.mpk'
Whole process took 0.53 seconds to complete
Removing temporal directory '/tmp/psocratescc_yQlavM'
Removing temporal filename '/tmp/psocratescc_DcZLO1'
[bsc@mppa-dev wavefront]$
```

In order to run the multibinary, programmers must use the *k1-jtag-runner* as shown in the rule "run" of the Makefile.

```
[bsc@mppa-dev wavefront]$ make run
k1-jtag-runner --multibinary io_elf.mpk --exec-multibin=IODDR0:io_elf
IO0@0.0: 00:  DIM    N  #tasks   Result
IO0@0.0: 00:   256   4      16  3057647614
```