PoCC

The Polyhedral Compiler Collection package
Edition 0.3, for PoCC 1.0-rc3
February 8th 2011

Louis-Noël Pouchet

This manual is dedicated to PoCC version 1.0-rc3, a flexible source-to-source compiler in the polyhedral model.

Table of Contents

1 Introduction

PoCC is a full compiler for polyhedral optimization. It leverages most of the state-of-the-art polyhedral tools in the public domain, resulting in a flexible and powerful compiler.

Communication: three groups are available for subscription

- pocc-announces, to receive announces about releases of the software.
- pocc-users, to discuss any matter about how to use PoCC.
- pocc-dev, to discuss any matter related to the software's implementation, bug fixes, desired features, etc...

Please contact the author directly for any question.

API Documentation: There is a Doxygen documentation of the API available in `doc/htmldoc.tar.gz`.

License: PoCC is released under the terms of the GNU Lesser General Public License version 2.1 and later. The software it uses is released under the terms of the GNU GPL v2 or later, and GNU Lesser GPL v2 or later.

Acknowledgments

PoCC could not exist without the wonderful software it includes, and its authors diserve most of the credit.

- CLooG has been developped by Cédric Bastoul and Sven Verdoolaege
- Clan and Candl have been developped by Cédric Bastoul and Louis-Noël Pouchet
- PLuTo has been developped by Uday Bondhugula
- LetSee has been developped by Louis-Noël Pouchet
- ISL has been developped by Sven Verdoolaege
- PIPLib has been developped by Paul Feautrier and Cédric Bastoul
- PolyLib has been developped mainly by Doran Wilde, Bart Kienhuis, Vincent Loechner, Tanguy Risset and Sven Verdoolaege
- FM has been developped by Louis-Noël Pouchet

2 PoCC Installation

PoCC requires `perl` to be installed in order to work properly. It also requires of course a Shell and a working C compiler.

The development version of PoCC (currently the only one being distributed) additionally requires the standard GNU development tools: `bison`, `flex`, reasonably recent version of `libtool` (1.5.22 at least, beware of the libtool/glibtool issue on Mac OSX), `autoconf 2.60` or later, `automake 1.10` or later, `doxygen`, and both a `subversion` and `git` client.

To install PoCC and all its components, simply do the following:

```
$> tar xzf pocc-1.0-rc3.tar.gz
$> cd pocc-1.0-rc3
$> ./install.sh
```

This will configure and build the development version of PoCC on your system. PoCC is not aimed at being installed on the computer in one of the standard locations (`/usr/bin` for instance). Instead, add the `bin` directory to your PATH variable to be able to use PoCC.

```
$> export PATH=‘pwd‘/bin:$PATH
```

In the current development version, we resorted to ship a standard version of GMP. This is required by ISL, which is required by CLooG. By default, the configuration of GMP is automatic. To change this, for instance because PoCC is being installed on a fake 64bits operating system such as Mac OS 10.4, or using a 32-bit compiler on a 64 bit architecture such as in Mac OS 10.5, modify the variable `GMP_ABI_FORCE` at the beginning of `install.sh`. For instance to force 32bit compilation of GMP set `GMP_ABI_FORCE="ABI=32"`.

Finally, to test that the compiler works, use the `gemver.c` file at the root of the archive for a test run.

```
$> pocc gemver.c
```

It is **strongly recommended** to run the `make check` command, to ensure the compiler works

properly on sanity benchmarks at least.

```
$> make check
```

3 The PoCC Compiler

This chapter describes briefly the possibilities of the PoCC compiler.

3.1 Input Code

To delimit which part of the code is going to be compiled by the polyhedral optimizer(s), it is required to delimit the code with `#pragma scop` and `#pragma endscop`. For the *base* and *devel* configuration of PoCC, the code fragment inside the pragmas must be a regular static control part: a consecutive set of statements with only for loops, where loop bounds, `if` statement conditionals and array accesses are affine functions of the iterators and the global parameters. The ternary operator can be used to provide data-dependent code, where the control will be over-approximated to both clauses being executed for all possible executions.

```
#pragma scop
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++) {
            A[i][j] = A[i][j] + u1[i]*v1[j]
            if (N - i > 2)
               A[i][j] -= 2;
         }
     res = A[0][0] == 0 ? u1[i] : v1[j];
#pragma endscop
```

For a better treatment of irregular program, consider the *irregular* configuration of PoCC:

```
$> pocc-util alternate irregular
```

This will rebuild PoCC with the irregular modules (highly experimental). To restore to the initial configuration, do:

```
$> pocc-util alternate base
```

Or, if you are a development user with access to the SVN of ALCHEMY at Inria:

```
$> pocc-util alternate devel
```

3.2 Available Software

PoCC leverages several GNU tools for polyhedral compilation. Namely:

- Clan, the Chunky loop analyzer, to extract a polyhedral intermediate representation from the source code.
- Candl, the Chunky analyzer for dependences in loops, to compute polyhedral dependences from the polyhedral IR.
- LetSee, the Legal transformation Space explorator, for iterative compilation using affine multidimensional schedules
- PLuTo, an automatic parallelizer and locality optimizer for multicores, for powerful optimization with tiling and parallelism in the polyhedral model
- CLooG, the Chunky Loop Generator, to generate syntactic code from the polyhedral representation
- ISL, the Integer Set Library
- PIPLib, the Parametric Integer Programming Library
- PolyLib, the Polyhedral Library
- FM, the Fourier-Motzkin library

3.3 Available Options

These are the available options for PoCC.

3.3.1 General Options

- `-h`, `--help` Print this help
- `-v`, `--version` Print version information
- `-o`, `--output <arg>` Output file [filename.pocc.c]
- `-d`, `--delete-files` Delete files previously generated by PoCC [off]
- `--bounded-ctxt` Parser: bound all global parameters >= -1 [off]
- `--verbose` Verbose output [off]
- `--quiet` Minimal output [off]

3.3.2 ScopLib Input/Output Options

- `--read-scop` Parser: read SCoP file instead of C file
- `--output-scop` Output scoplib file to filename.pocc.scop
- `--cloogify-scheds` Create CLooG-compatible schedules in the scop. When invoked in conjunction of –ouput-scop, the schedules in the generated scop can be directly given as an input to CLooG.

3.3.3 Dependence Analysis

- `--no-candl` Dependence analysis: don't run candl [off]
- `--pluto-ext-candl` PLuTo: Read dependences from SCoP [off]
- `--candl-dep-isl-simp` Dependence analysis: simplify dependence polyhedra with ISL [off]

3.3.4 Code Generation Options

- `-n`, `--no-codegen` Do not generate code [off]
- `--cloog-cloogf <arg>` CLooG: first level to scan [1]
- `--cloog-cloogl <arg>` CLooG: last level to scan [-1]
- `--codegen-timercode` Codegen: insert timer code [off]
- `--codegen-timer-asm` Codegen: insert ASM timer code [off]
- `-c`, `--compile` Compile program with C compiler [off]
- `--compile-cmd <arg>` Compilation command [gcc -O3 -lm]
- `--run-cmd-args <arg>` Program execution arguments []
- `--prog-timeout` Timeout for compilation and execution of a single version, in second [unliminted]

For instance, to perform a full compilation and create a binary, type:

```
$> pocc --compile gemver.c -o gemver
```

3.3.5 Optimization Options

About LetSee:

- `-l`, `--letsee` Optimize with LetSee [off]
- `--letsee-space <arg>` search space: [precut], schedule
- `--letsee-walk <arg>` traversal heuristic: [exhaust], random, skip, m1, dh
- `--letsee-dry-run` Don't compile and run programs for iterative search [off]
- `--letsee-bounds` search space bounds [-1,1,-1,1,-1,1]
- `--letsee-normspace` normalize search space [off]
- `--letsee-mode-m1 <arg>` scheme for M1 traversal [i+p,i,0]
- `--letsee-rtries <arg>` number of random draws [50]
- `--letsee-prune-precut` prune precut space
- `--letsee-backtrack` allow bactracking in schedule mode

About PLuTo:

- `-p`, `--pluto` Optimize with PLuTo [off]
- `--pluto-parallel` OpenMP parallelization [off]
- `--pluto-tile` polyhedral tiling [off]
- `--pluto-l2tile` perform L2 tiling [off]
- `--pluto-fuse <arg>` fusion heuristic: nmaxfuse, [smartfuse], nofuse

- `--pluto-unroll` unroll loops [off]
- `--pluto-ufactor <arg>` unrolling factor [4]
- `--pluto-polyunroll` polyhedral unrolling [off]
- `--pluto-prevector` perform prevectorization [off]
- `--pluto-multipipe` multipipe [off]
- `--pluto-rar` consider RAR dependences [off]
- `--pluto-lastwriter` perform lastwriter dep. simp. [off]
- `--pluto-scalpriv` perform scalar privatization [off]
- `--pluto-bee` use Bee [off]
- `--pluto-quiet` be quiet [off]
- `--pluto-ft` ft [off]
- `--pluto-lt` lt [off]
- `--pluto-tile-scat` PLuTo: Embed tiling inside scatterings [off]

For instance, to perform tiling with the Tiling Hyperplane method of Bondhugula et al, type:

```
$> pocc --pluto-tile gemver.c
```

As a more complicated example, let us consider the precut (aka fusion structure) method. To compute (without executing) the number of points in the search space for `gemver.c` type:

```
$> pocc ---letsee --verbose --no-codegen  gemver.c

[...]
[LetSee] Generated 0 fusion structures from 8 schedules
[...]
```

To perform 5 random tests in the space of legal fusion structures for `gemver.c`, together with tiling and parallelization activated, type:

```
$> pocc --compile-cmd "gcc -O3 -fopenmp" --letsee-walk random \
--letsee-rtries 5 --pluto-tile --pluto-parallel --codegen-timercode gemver.c
```

4 Developping with PoCC

This chapter describes briefly how to program in PoCC. As a kicker example, we will create a new mode for PoCC (such as base, devel and irregular) named *cuda*. We will then add a new module into the toolchain, named *cudascheduler*. We also discuss at the end various tips and tricks for the compilation toolchain.

4.1 Create a new Mode for PoCC

To create a new mode for pocc, named cuda, one can do the following. The idea is to create a configuration based on the **devel** mode. At start, we simply duplicate this configuration.

```
$> mkdir config/cuda
$> cp config/devel/* config/cuda
$> emacs config/cuda/driver.cfg
   change the line "configuration: devel" to "configuration: cuda"
$> pocc-util alternate cuda
```

The last step, `pocc-util alternate cuda` recompiles everything, and should work fluently.

4.2 Create and use a new module

We now want to create a new module, *cudascheduler*. This module will compute a transformation for the program in the form of an affine schedule. Hence it is categorized as an optimizer.

4.2.1 Create the new module from scratch

To comply with standard GNU application template, we recommend to use `autoskelet-0.1` (can be downloaded from L.N. Pouchet website). Download and extract the archive, and go to the directory autoskelet-0.1. Then do the following.

- Create a NOTICE file for the project, `NOTICE.cuda` for instance. This will be the header for all generated files. Beware to use the string `FILENAME` as is, it will be automatically substituted by the actual filename during the project generation. Inspire yourself from `NOTICE` for GPL v2 projects and `NOTICE.LESSER` for LGPL v3 projects.
- Create the project:

```
$> ./make_project.sh cudascheduler /full/path/to/pocc/optimizers/cudascheduler
templates/project_bin_static_shared NOTICE.cuda
```

- Perform final edits (`AUTHORS`, etc.)
- We heavily recommand to store the module on a SVN or Git repository. Currently PoCC supports only module stored as svn or git repositories, and as `.tar.gz` archives on a local location

4.2.2 Insert the new module in the toolchain

First, add the information about the configuration of the new module into the `config/cuda/configure.cfg` file, which stores the specifics for building all modules. A working template is the following, which adds the module `cudascheduler` located on a SVN repository, and depending on the `scoplib`. Note we do not have yet implemented the support of the `--with-scoplib` option in our module, still we can proceed.

```
[cudascheduler]
category: optimizers
module: cudascheduler
location: https://alchemy.futurs.inria.fr/svn/users/....
retrieve-method: svn
bootstrap: bootstrap.sh
configure: --with-scoplib=$poccsrcdir/ir/install-scoplib
make-depends: scoplib
```

Refer to `configure.cfg` for more information. `$poccsrcdir` is automatically substituted with the absolute path of the PoCC compiler. Also, a module named `xxx` is automatically installed in the directory `install-xxx`.

Second, inform the driver that it needs the new module to be part of the build chain, by editing `config/cuda/driver.cfg` and adding `cudascheduler` in the `software-versions` list. Finally, recompile PoCC.

```
$> pocc-util alternate cuda
```

At this stage, it should still work fluently.

4.2.3 Editing the new module

Now you can start your modifications of the project. For instance, adding the code to manage the `--with-scoplib` option for the configure script is a good idea. One could for instance refer to `generators/cloog/configure.in` file, or directly add the following code fragment into `optimizers/cudascheduler/configure.ac`.

```
dnl /*********************************************************************
dnl  *                        Where is ScopLib?                          *
dnl  *********************************************************************/

dnl Offer --with-scoplib.
  AC_ARG_WITH(scoplib,
      AC_HELP_STRING([--with-scoplib=DIR],
                             [DIR Location of ScopLib package]),
              [with_scoplib=$withval;
       CPPFLAGS="${CPPFLAGS} -I$withval/include";
       LDFLAGS="${LDFLAGS} -L$withval/lib"
      ],
              [with_scoplib=check])
dnl Check for scoplib existence.
  AS_IF([test "x$with_scoplib" != xno],
[AC_CHECK_LIB([scoplib], [scoplib_scop_read],
 [LIBS="-lscoplib $LIBS";
 DEFINE_HAS_SCOPLIB_LIB="-DCUDASCHED_SUPPORTS_SCOPLIB"
 ],
         [DEFINE_HAS_SCOPLIB_LIB=""
     if test "x$with_scoplib" != xcheck; then
           AC_MSG_FAILURE([Test for ScopLib failed. \
Use --with-scoplib to specify libscoplib path.])
          fi
         ])
])
AC_SUBST(DEFINE_HAS_SCOPLIB_LIB)
```

You do not longer need to run pocc-util alternate to compile, just do

```
$> cd optimizers/cudascheduler && make install; cd -
```

The `make install` command directly into the `cudascheduler` module should bootstrap/reconfigure/make/install automatically when needed.

4.2.4 Activating the new pass into PoCC

First, create an option in the option list.

- Edit `driver/pocc/include/pocc/options.h` and add a variable storing the option value.
- Edit `driver/pocc/options.c` and add management (initialization / cleaning) code for this new option

- Edit `driver/src/options.h` and add a new defined value for the new option, for instance `#define POCC_OPT_CUDASCHEDULER 43`. Don't forget to increment the `POCC_NB_OPTS` define.
- Edit `driver/src/options.c` and update the structure `opts` at the beginning of the file. Beware to insert the new option management **at the position defined in** `options.h`. Add the code to read the option value in the `pocc_getopts` function.

Second, update build information with the new module, and create a new file for the new pass.

- Edit `driver/pocc/Makefile.am` and update the `INCLUDE` variable with the following:

```
-I$(top_builddir)/optimizers/install-cudascheduler/include \
-I$(top_srcdir)/optimizers/install-cudascheduler/include \
```

  Also edit the `libpocc_la_LIBADD` variable and add

```
$(top_builddir)/optimizers/install-cudascheduler/lib/libcudascheduler.la \
```

- Create a new file to manage the new pass, for instance `driver/pocc/driver-cudascheduler.c`. Do not forget to add this new file in the `libpocc_la_SOURCES` variable of `driver/pocc/Makefile.am`. You are strongly encouraged to add also a dedicated header file `driver/pocc/include/pocc/driver-cudascheduler.c`.
- Call the module exported function the way you want...

Lastly, update PoCC driver to call the new pass. Don't forget to use the option we have created before.

- Edit `driver/src/pocc.c` and call the exported function of your driver. Don't forget to add the header file of this new pass into the list of included files of `driver/src/pocc.c`.

And you are done. Enjoy your development now!

4.3 Development tips and tricks

You want to change the configure script options for cudascheduler:

```
$> emacs config/cuda/configure.cfg, do your changes
$> rm optimizers/cudascheduler/configure
$> pocc-util alternate cuda
```

You want to build the whole project (but no modif were made to config/cuda/*) in the safest fashion:

```
$> pocc-util alternate cuda
```

You have modified something in `optimizers/cudascheduler`, but not elsewhere

```
$> cd optimizers/cudascheduler && make install; cd -
```

You have modified something in `optimizers/cudascheduler` and in `driver/` only:

```
$> cd optimizers/cudascheduler && make install; cd -
$> make install
```

You have modified some modules and/or the driver, but not the configuration:

```
$> pocc-util buildall
```

You want to work only on one module (possibly just added in `mode/configure.cfg`) and use the PoCC tools to checkout and compile it:

```
$> pocc-util checkout mymodule
$> pocc-util make mymodule
```

5 Troubleshoot

5.1 Build error during the first installation

The most common problem arises with the compilation of GMP. Unfortunately the problem occurs only when reaching the compilation of CLooG, which makes it hard to diagnose.

It has been reported that for Mac OS 10.4, and for Mac OS 10.5 and later *using the 32-bit native Apple-GCC compiler* that the compilation of GMP must be forced to 32 bits. To do do:

- Delete the `math/external/gmp-4.3.1` directory
- Edit `install.sh` and set `GMP_ABI_FORCE="ABI=32"`
- Re-run install.sh

It has also been reported once a problem of a missing include `<gmp.h>`. If you face this problem a quick-and-dirty solution is to copy `math/external/install/include/gmp.h` into `generators/install-cloog/include`. Please also report the problem to the author louis-noel.pouchet@inria.fr.

5.2 Other problems

The most usual problem comes from software dependences not met, in particular software versions. Refer to the beginning of this manual to ensure you have recent-enough software version.

Simply contact the author for the moment: louis-noel.pouchet@inria.fr

Please provide the output of the following commands:

```
$> uname -a
$> gcc -v
$> libtool --version
$> automake --version
$> autoconf --version
$> flex --version
$> bison --version
$> git --version
$> svn --version
```

6 References

[1] Cédric Bastoul. Code Generation in the Polyhedral Model Is Easier Than You Think. In *IEEE International Conference on Parallel Architecture and Compilation Techniques (PACT'04)*, pp 7–16, Juan-les-Pins, France, September 2004.

[2] Uday Bondhugula and Albert Hartono and J. Ramanujam and P. Sadayappan. A Practical Automatic Polyhedral Program Optimization System. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'08)*, ACM Press, Tucson, Arizona, June 2008.

[3] Paul Feautrier. Parametric Integer Programming. In *RAIRO Recherche opérationnelle*, Vol 22, num 3, pp 243–268, 1998.

[4] Louis-Noël Pouchet, Cédric Bastoul, Albert Cohen and John Cavazos. Iterative optimization in the polyhedral model: Part II, multidimensional time. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'08)*, pp 90–100, ACM Press, Tucson, Arizona, June 2008.

[5] Doran Wilde. A library for doing polyhedral operations. In *Journal of Parallel Algorithms and Applications*, Vol 15, pp 137–166, 2000.