Tpips: command line interpreter

Corinne Ancourt Fabien Coelho Guillaume Oget

March 2, 2016 *(revision 23065)*

You can get a printable version of this document on `http://www.cri.ensmp.fr/pips/tpips-user-manual.htdoc/tpips-user-manual.pdf` and a HTML version on `http://www.cri.ensmp.fr/pips/tpips-user-manual.htdoc`.

Contents

1 Introduction

`tpips` is the **line interface** and scripting language of the PIPS project. All PIPS functionalities are available, with on-line help and automatic completion. `tpips` is more efficient than the `pips` based scripts (`Init`, `Display`,...) because the database is not open, read, written and closed for each command. `tpips` is less user-friendly than `wpips` and `epips`[1], although it provides the unique opportunity to apply a transformation or display some information on several modules with one command, using pipsmake metavariables, `%CALLERS` and `%CALLEES`, or environment user defined variables such as `$MODULES` (see Example 4.2). The `jpips` future user interface will provide these `pips`/`tpips` functionalities too.
`tpips` is recommended for large benchmarks and experiments, when analysis and optimization results do not require interactive working. `tpips` is used for the validation of each new PIPS version.
`tpips` can be used to automatically replay a manual `wpips` or `tpips` session by using `logfile_to_tpips`. The `logfile_to_tpips` command derives a `tpips` script from a logfile in a workspace (see Example 4.2). A PIPS beginner should start with a window interface, `wpips` or `epips`[2], and not with `tpips`. But the simple tools of section 10 can be useful for simple tasks.
Access to Unix Shell and to PIPS properties is provided by `tpips`. PIPS properties can be modified from `tpips` , and only from `tpips` during a PIPS session (see Example 4.3). On the other hand, `wpips` reads at the beginning of the session the default properties and the new ones (re)defined in the `properties.rc` file, but no change can be done afterwards. Resetting properties during a session can be used to rebuild an application when some modules have been updated by PIPS, e.g. after partial evaluation.
The `tpips` commands are listed on-line by the help command. Possible arguments are completed or suggested by automatic completion (TAB and TAB/TAB). A command may be spread on several lines by guarding the line feed characters by backslashes.

2 Syntax

Different functions can be activated with the `tpips` commands. We decompose these functions in those related to the *workspace*, the *analysis* and *transformation* requests, the PIPS *properties* and *options* and the the environment. Functions and their arguments are presented in the next subsections.

2.1 Workspace

The commands for workspaces are: `create` a new workspace (see `pips` command `Init`), `open` an existing workspace, `close` the current workspace, and `delete` the current workspace or a specific one (see `pips` command `Delete`).
These commands can be used several times during a session or within a `tpips` script file.

[1] Note that `wpips` and `epips` don't work anymore right now because a deprecated GUI library, so `tpips` is definitively the way to go...

[2] If they work...

create <**workspace-name**> <**file-name**> creates and opens a new workspace from a source file — or set of source files

open <**workspace-name**> opens an existing workspace

close <**workspace-name**> closes an opened workspace

delete <**workspace-name**> deletes a closed workspace

quit quits `tpips`

2.2 Requests

To apply a transformation or request a particular resource, use the following `tpips` commands.
For a list of the available resources and their descriptions, have a look to `http://www.cri.ensmp.fr/pips/pipsmake-rc.htdoc` or `http://www.cri.ensmp.fr/pips/pipsmake-rc.htdoc/pipsmake-rc.pdf`.

apply <**rule-descriptor**> applies a specified rule and produces the associated resources.

capply <**rule-descriptor**> applies concurrently a specified rule on the different *OWNER*s of the <rule-descriptor> and produces the associated resources (see Example 4.4).

display <**resource-descriptor**> makes a printable resource (if it does not exist) and prints it out. A printed resource is a ACSII file with the `_FILE` extension.

make <**resource-descriptor**> makes a resource according to default rules

Generic example:

A resource, i.e. any piece of information computed by PIPS, can be required by command:

```
make␣<resource>
```

or

```
make␣<resource[owner]>
```

which use interproceduraly and recursively default rules (see `pips` command `Build`). Or it is produced by a specific rule:

```
apply <rule>
```

which will require all its resource arguments by recursively calling make (see `pips` command `Perform`).
The owner of a resource is the module it is attached to. By default, the owner is assumed to be the current module.
File resources can be displayed on screen:

```
display <printable_resource>
```

as with the `pips Display` script.

2.3 PIPS properties and options

For a list of the available properties and their descriptions, have a look to `http://www.cri.ensmp.fr/pips/properties-rc.htdoc` or `http://www.cri.ensmp.fr/pips/properties-rc.htdoc/properties-rc.pdf`.

activate <**rule-name**> activates a rule. Several rules can be used to produce a resource. At any time one of these rules is the default rule.

module <**module-name**> specifies the current module to work on.

getproperty <**property-name**> prints the value of a PIPS property.

setproperty <**property-name**> <**value**> sets the value of a property — after type check. Possible types are: boolean, int and string.

checkpoint <**number**> checkpoints the current workspace every <number> pipsmake operations.

Generic example:

A specific current module is selected as current module with:

```
module MODULE_NAME
```

When several rules, i.e. functions, can be used to compute the same resource, for instance a parallel version of a module, the rule selection command is:

```
activate <rule>
```

It is equivalent to the `pips Select` command.

2.4 Environment

The `tpips` Unix-related commands are: `cd`, `setenv`, `getenv`, `echo`, and `shell`.

shell <**sh-command**> <**parameters**> Executes the line as a shell command. With no arguments, run an inferior shell;

source <**filenames...**> reads `tpips` commands from files;

cd <**directory-name**> changes the default directory;

echo <**string**> prints <string>;

getenv <**variable-name**> prints the value of <variable-name> (equivalent to `echo ${<variable-name>}`);

setenv <**variable-name**> = <**value**> sets <value> to <variable-name>;

help <**command**> prints out general help or detailed help about an item;

info <**name**> print information about <name>. Here <name> can be the `module`, `modules`, `workspace` or `directory` words.

2.5 Parameters

The parameters of the different commands are :

<**file-name**> Unix path and file name.

Shell syntax, such as `*.f`, can be used when `s` option is specified or `TPIPS_IS_A_SHELL` variable is set;

<**workspace-name**> Unix file name — no path;

<**module-name**> Fortran function name in capital letters or a C function name;

<**rule-name**> rule name (see Section 5);

<**resource-descriptor**> It is a `resource[OWNER]` expression: a pipsmake resource name to be computed for each module associated to OWNER (see Sections 6 and 7);

<**rule-descriptor**> It is a `rulename[OWNER]` expression: a pipsmake instantiated rule to be applied on each module associated to OWNER (see Section 5);

the OWNER — used in the previous <rule-descriptor> and <resource-descriptor> — can take these formats:

module-name the target module name;

%PROGRAM the whole program (for global resource);

%MAIN the main module of the program;

%MODULE the current module;

${MODULES} the modules associated to the Shell variable `MODULES`. The different names are SPACE or/and TAB separated;

%ALL All workspace modules, including C compilation units[3];

%ALLFUNC All workspace functions, excluding C compilation units;

%CALLEES All the callees of the current selected module;

%CALLERS All the callers of the current selected module.

If no OWNER is specified, then the current module is assumed. For generic owners, a sign (%) has been added to differentiate them from existing module names and environment variables.

3 Simple session

A simple `tpips` session is made of a few steps. First, a workspace must be created:

```
create work source1.f source2.f
```

[3] A PIPS compilation unit is associated to each C user source file. it contains all the initial declarations, but the function definitions.

and a subdirectory of the current directory, here `work.database`, is created to store and retrieve code and information computed by PIPS about subroutines and functions found in `source1.f` or `source2.f`.
Then, you have to select a module, let say `FOO`, by typing:

```
module FOO
```

unless there is only one function or subroutine in the Fortran file(s) which were passed to the create command, in which case it is automatically selected.
To keep it simple the first time, you should then try to display the sequential code of routine `FOO` by typing:

```
display PRINTED_FILE
```

Other resource names let you visualize the parallelized code (`PARALLELPRINTED_FILE`), or the call graph (`CALLGRAPH_FILE`)... Names for printable resources usually contain the string `FILE`.
The standard command to leave `tpips` is `quit` but a control-D or end-of-file condition are valid too.
Interactive analysis of module `MATMUL` from source code `pgmsrc.f`, in workspace `pgm`:

```
$ tpips
tpips > create pgm pgm-src.f

tpips > module MATMUL

tpips > display PARALLELPRINTED_FILE

tpips > quit
```

4 Examples

4.1 Call Graph

To visualize the call graph of the application, the following tpips session can be executed. The call graph of each application routine is displayed.
This example presents a use of:

- environment variable: `setenv PIPS_SRCPATH SRC`
- shell path file name expression that will be expanded: `*.f`,
- a <resource-descriptor> : `CALLGRAPH_FILE[%ALL]`

This `tpips` example is useful to test PIPS parsing on each new benchmark.

```
setenv PIPS_SRCPATH SRC      # initiates the source file directory to SRC

create wc *.f                 # creates the wc workspace for the
```

```
                                # Fortran files of the  PIPS_SRCPATH
                                # directory

display CALLGRAPH_FILE[%ALL]  # CONTROL GRAPH COMPUTATION for each routine

close
quit
```

4.2 Dependences

A programmer that would like to see the routines dependence graph to parallelize interprocedurally the application can executed the following `tpips` commands.
This example presents some PIPS analyses and use of

- module selection: `module TEST`
- <rule-name>: `MUST_REGIONS`, `REGION_CHAINS`
- `logfile_to_tpips` command

```
sh>tpips                              # run tpips
tpips> create wfoo essai.f            # create a workspace for the program essai.f
tpips> module TEST                    # select the default module TEST
tpips> display CALLGRAPH_FILE         # display the sub-callgraph for the
                                         module TEST
tpips> make PROPER_EFFECTS[%ALL]      # compute proper effects for every
                                         modules
tpips> activate MUST_REGIONS          # select the rule must_regions instead
                                         of the default rule may_regions
tpips> activate REGION_CHAINS         # select the rule regions_chains for
                                         computing Use-Def chains in essai.f
tpips> display DG_FILE[%ALL]          # display all the dependance graphs
tpips> close wfoo                     # close the workspace
tpips> quit                           # quit tpips
```

To store this simple session in a `tpips` file, use the command:

```
logfile_to_tpips wfoo > foo.tpips
```

that generates the following `tpips` commands in the file `foo.tpips`.

```
#
# file generated by logfile_to_tpips for workspace wfoo
#
create wfoo essai.f
module TEST
display CALLGRAPH_FILE[TEST]
display PROPER_EFFECTS[TEST]
activate MUST_REGIONS
```

```
activate REGION_CHAINS
display DG_FILE[TEST]
close
quit
# EOF
```

ote that expression `%ALL` has been expanded by all program routines, in that example there is only one `TEST` routine in `essai.f`.

4.3 Transformations

The following transformations are designed for application optimization. Specialization and code reduction eliminating useless instructions and declarations are performed.
This example presents some PIPS transformations and use of

- program restructuration, cloning, dead code elimination, useless declaration elimination, and code regeneration;
- pips property: `PREPROCESSOR_MISSING_FILE_HANDLING "generate"`;
- transformation application : `apply CLONE_ON_ARGUMENT[FUNCT]`.
- <rule-name> : `PRINT_CODE`...
- <rule-descriptor>: `SUPPRESS_DEAD_CODE[${modules}]`, `UNSPLIT[%PROGRAM]`...
- <resource-descriptor> : `PRINTED_FILE[FUNCT]`, `CALLGRAPH_FILE[%ALL]`...

```
# Delete Workspace clone in case  it already exists

delete clone

setenv PIPS_SRCPATH SRC

# Stop as soon as the first user error occurs

setproperty ABORT_ON_USER_ERROR TRUE

# Generates missing files  and routines

setproperty PREPROCESSOR_MISSING_FILE_HANDLING "generate"

# Substitute Fortran statement functions

setproperty PARSER_EXPAND_STATEMENT_FUNCTIONS TRUE

# Restructures the program to eliminate as many GOTO as possible

setproperty UNSPAGHETTIFY_TEST_RESTRUCTURING=TRUE
setproperty UNSPAGHETTIFY_RECURSIVE_DECOMPOSITION=TRUE
```

```
#
# prettyprinter settings

setproperty PRETTYPRINT_ALL_DECLARATIONS TRUE

# creates the workspace

create clone *.f


echo # initial version of FUNCT

display PRINTED_FILE[FUNCT]

#
# just to insure that all routines are generated.

make CALLGRAPH_FILE[%ALL]

#
# let us clone FUNCT on the second argument.
# no interprocedural analysis is needed.

setproperty TRANSFORMATION_CLONE_ON_ARGUMENT 2
capply CLONE_ON_ARGUMENT[FUNCT]

# The clone transformation generates 4 clones
# because there are 4 different values for the second
# argument of FUNCT

setenv modules "FUNCT_0 FUNCT_1 FUNCT_2 FUNCT_3"

#
# suppress dead code must be applied twice...
# eliminate dead code after cloning

apply SUPPRESS_DEAD_CODE[${modules}]
apply SUPPRESS_DEAD_CODE[${modules}]

# Eliminates useless declaration after cloning and dead code
# suppression

apply CLEAN_DECLARATIONS[FUNCT ${modules}]

echo # after cloning, dead code elimination and declarations cleaning

make  PRINTED_FILE[${modules}]

#
# regenerates the source files with calls to cloned functions only
```

```
#

activate PRINT_CODE
apply UNSPLIT[%PROGRAM]

close
```

4.4 Analyses

This example presents some classical PIPS analyzes:

- computation of preconditions;
- transformers;
- effects;
- regions

and a *concurrent* transformation application.
`capply PARTIAL_EVAL[%ALL]` applies a partial evaluation on all routines concurrently without any verification process between two transformations. The partial evaluation of a routine does not modify other routine predicates, even if the code (one routine) has been modified by the transformation. It is not necessary to ask for a sequential application that will impose the analysis and re-computation of all routine predicates between two partial evaluations.

```
create foo *.f

echo
echo CALL_GRAPH[%ALL] Computation
echo

display CALLGRAPH_FILE[%ALL]

# ask interprocedural information

activate PRECONDITIONS_INTER_FULL
activate TRANSFORMERS_INTER_FULL

module FUNCT

capply PARTIAL_EVAL[%ALL]

#echo
#echo PRINT_CODE_PROPER_EFFECTS Activation
#echo
#
#activate PRINT_CODE_PROPER_EFFECTS
#display PRINTED_FILE[%ALL]
#
#echo
```

```
#echo PRINT_CODE_CUMULATED_EFFECTS Activation
#echo
#
#activate PRINT_CODE_CUMULATED_EFFECTS
#display PRINTED_FILE[%ALL]

echo
echo PRINT_CODE_TRANSFORMERS Activation
echo


activate PRINT_CODE_TRANSFORMERS
display PRINTED_FILE[%ALL]

echo
echo PRINT_CODE_PRECONDITIONS Activation
echo

activate PRINT_CODE_PRECONDITIONS
display PRINTED_FILE[%ALL]

echo
echo ICFG_WITH_LOOPS_REGIONS
echo

activate PRINT_ICFG_WITH_LOOPS_REGIONS
display ICFG_FILE[%ALL]

echo
echo REGIONS Computation
echo

activate MUST_REGIONS
activate PRINT_CODE_REGIONS
display PRINTED_FILE[%ALL]


close
quit
```

4.5 Graphical Graph View

This example presents a call to a graphical call graph view.

```
# resize the entities table. Useful for  large benchmarks.

setenv NEWGEN_MAX_TABULATED_ELEMENTS 150000
setenv MAINROUTINE  TOTO

delete DVCG
```

```
create DVCG *.f


echo
echo CALLGRAPH_FILE[${MAINROUTINE}] Computation
echo


make CALLGRAPH_FILE[${MAINROUTINE}]


echo
echo DVCG_FILE[${MAINROUTINE}] Computation
echo

make DVCG_FILE[${MAINROUTINE}]

# type DaVinci DVCG.database/${MAINROUTINE}/${MAINROUTINE}.daVinci
# to  visualize

close
quit
```

4.6 Parallelization

This example presents some classical analyses and transformations designed to parallelize benchmarks, for instance the `adm` program of the Perfect club.

```
#
# Perfect club ADM benchmark.

echo Perfect/adm.f
#
delete adm
create adm adm.f  cputim.f elapse.f

setproperty UNSPAGHETTIFY_TEST_RESTRUCTURING=TRUE
setproperty UNSPAGHETTIFY_RECURSIVE_DECOMPOSITION=TRUE
setproperty PARALLELIZATION_STATISTICS=TRUE

echo ADM scalar privatization...

capply PRIVATIZE_MODULE[%ALL]


activate PRECONDITIONS_INTER_FULL
activate TRANSFORMERS_INTER_FULL

activate RICE_SEMANTICS_DEPENDENCE_GRAPH
```

```
apply PARTIAL_EVAL[%ALL]

echo ADM parallelization...
make PARALLELPRINTED_FILE[%ALL]

close
quit
```

5 Rules — Pips options

Rules are used to compute resources. Several rules can be activated to compute a single resource. For instance a parallel version of a module may be computed from the interprocedural precondition (`PRECONDITIONS_INTER_FULL`), interprocedural transformer (`TRANSFORMERS_INTER_FULL`) and accurate dependencies (`RICE_SEMANTICS_DEPENDENCE_GRAPH`). Rules are selected by `activate` or selected and applied by `apply` and `capply`. More rules and more information on each particular rule are presented in the pipsmake documentation, `http://www.cri.ensmp.fr/pips/pipsmake-rc.htdoc` or `http://www.cri.ensmp.fr/pips/pipsmake-rc.htdoc/pipsmake-rc.pdf`.

```
####␣ANALYSES␣####

ATOMIC_CHAINS
CONTINUATION_CONDITIONS
FLINTER
PROPER_REFERENCES

###␣COMPLEXITITY

ANY_COMPLEXITIES
FP_COMPLEXITIES
UNIFORM_COMPLEXITIES
SUMMARY_COMPLEXITY

###␣PRIVATIZATION

ARRAY_PRIVATIZER
ARRAY_SECTION_PRIVATIZER
DECLARATIONS_PRIVATIZER
PRIVATIZE_MODULE

###␣EFFECTS

CUMULATED_EFFECTS
CUMULATED_REDUCTIONS
CUMULATED_REFERENCES
IN_EFFECTS
IN_SUMMARY_EFFECTS
OUT_EFFECTS
```

```
OUT_SUMMARY_EFFECTS
PROPER_EFFECTS
SUMMARY_EFFECTS


###␣PRECONDITIONS

PRECONDITIONS_INTER_FAST
PRECONDITIONS_INTER_FULL
PRECONDITIONS_INTRA
SUMMARY_PRECONDITION


###␣TRANSFORMERS

SUMMARY_TRANSFORMER
TRANSFORMERS_INTER_FAST
TRANSFORMERS_INTER_FULL
TRANSFORMERS_INTRA_FAST
TRANSFORMERS_INTRA_FULL


###␣REGIONS

IN_OUT_REGIONS_CHAINS
IN_REGIONS
IN_SUMMARY_REGIONS
MAY_REGIONS
MUST_REGIONS
OUT_REGIONS
OUT_SUMMARY_REGIONS
REGION_CHAINS
SUMMARY_REGIONS

###␣CALLGRAPH

FULL_GRAPH_OF_CALLS
CALLGRAPH
GRAPH_OF_CALLS

###␣DEPENDENCES

RICE_ALL_DEPENDENCE
RICE_CRAY
RICE_DATA_DEPENDENCE
RICE_FAST_DEPENDENCE_GRAPH
RICE_FULL_DEPENDENCE_GRAPH
RICE_REGIONS_DEPENDENCE_GRAPH
RICE_SEMANTICS_DEPENDENCE_GRAPH
```

```
###␣REDUCTIONS

PROPER_REDUCTIONS
SUMMARY_REDUCTIONS


####␣TRANSFORMATIONS␣####

FORWARD_SUBSTITUTE
PARTIAL_EVAL

###␣RESTRUCTURATION

ATOMIZER
CLEAN_DECLARATIONS
CLONE
CLONE_ON_ARGUMENT
CLONE_SUBSTITUTE
RESTRUCTURE_CONTROL
SUPPRESS_DEAD_CODE
UNSPAGHETTIFY
UNSPLIT
USE_DEF_ELIMINATION

###␣LOOP␣TRANSFORMATION

DISTRIBUTER
FULL_UNROLL
LOOP_INTERCHANGE
LOOP_NORMALIZE
LOOP_REDUCTIONS
STRIP_MINE
UNROLL

###␣PARALLELIZATION

COARSE_GRAIN_PARALLELIZATION
NEST_PARALLELIZATION

####␣PRETTYPRINT␣####

###␣PRINT_CALL_GRAPH

PRINT_CALL_GRAPH
PRINT_CALL_GRAPH_WITH_COMPLEXITIES
PRINT_CALL_GRAPH_WITH_CUMULATED_EFFECTS
PRINT_CALL_GRAPH_WITH_IN_REGIONS
PRINT_CALL_GRAPH_WITH_OUT_REGIONS
PRINT_CALL_GRAPH_WITH_PRECONDITIONS
PRINT_CALL_GRAPH_WITH_PROPER_EFFECTS
```

```
PRINT_CALL_GRAPH_WITH_REGIONS
PRINT_CALL_GRAPH_WITH_TRANSFORMERS

###␣PRINT_CHAINS_GRAPH

PRINT_CHAINS_GRAPH

###␣PRINT_CODE␣␣WITH␣...

PRINT_CODE
PRINT_CODE_AS_A_GRAPH
PRINT_CODE_AS_A_GRAPH_COMPLEXITIES
PRINT_CODE_AS_A_GRAPH_CUMULATED_EFFECTS
PRINT_CODE_AS_A_GRAPH_IN_REGIONS
PRINT_CODE_AS_A_GRAPH_OUT_REGIONS
PRINT_CODE_AS_A_GRAPH_PRECONDITIONS
PRINT_CODE_AS_A_GRAPH_PROPER_EFFECTS
PRINT_CODE_AS_A_GRAPH_REGIONS
PRINT_CODE_AS_A_GRAPH_TRANSFORMERS
PRINT_CODE_COMPLEMENTARY_SECTIONS
PRINT_CODE_COMPLEXITIES
PRINT_CODE_CONTINUATION_CONDITIONS
PRINT_CODE_CUMULATED_EFFECTS
PRINT_CODE_CUMULATED_REDUCTIONS
PRINT_CODE_CUMULATED_REFERENCES
PRINT_CODE_IN_EFFECTS
PRINT_CODE_IN_REGIONS
PRINT_CODE_OUT_EFFECTS
PRINT_CODE_OUT_REGIONS
PRINT_CODE_PRECONDITIONS
PRINT_CODE_PRIVATIZED_REGIONS
PRINT_CODE_PROPER_EFFECTS
PRINT_CODE_PROPER_REDUCTIONS
PRINT_CODE_PROPER_REFERENCES
PRINT_CODE_PROPER_REGIONS
PRINT_CODE_REGIONS
PRINT_CODE_STATIC_CONTROL
PRINT_CODE_TRANSFORMERS

###␣PRINT_DEPENDENCE_GRAPH

PRINT_EFFECTIVE_DEPENDENCE_GRAPH
PRINT_LOOP_CARRIED_DEPENDENCE_GRAPH
PRINT_WHOLE_DEPENDENCE_GRAPH

###␣PRINT_ICFG

PRINT_ICFG
PRINT_ICFG_WITH_COMPLEXITIES
PRINT_ICFG_WITH_CONTROL
```

PRINT_ICFG_WITH_CONTROL_COMPLEXITIES
PRINT_ICFG_WITH_CONTROL_CUMULATED_EFFECTS
PRINT_ICFG_WITH_CONTROL_IN_REGIONS
PRINT_ICFG_WITH_CONTROL_OUT_REGIONS
PRINT_ICFG_WITH_CONTROL_PRECONDITIONS
PRINT_ICFG_WITH_CONTROL_PROPER_EFFECTS
PRINT_ICFG_WITH_CONTROL_REGIONS
PRINT_ICFG_WITH_CONTROL_TRANSFORMERS
PRINT_ICFG_WITH_CUMULATED_EFFECTS
PRINT_ICFG_WITH_IN_REGIONS
PRINT_ICFG_WITH_LOOPS
PRINT_ICFG_WITH_LOOPS_COMPLEXITIES
PRINT_ICFG_WITH_LOOPS_CUMULATED_EFFECTS
PRINT_ICFG_WITH_LOOPS_IN_REGIONS
PRINT_ICFG_WITH_LOOPS_OUT_REGIONS
PRINT_ICFG_WITH_LOOPS_PRECONDITIONS
PRINT_ICFG_WITH_LOOPS_PROPER_EFFECTS
PRINT_ICFG_WITH_LOOPS_REGIONS
PRINT_ICFG_WITH_LOOPS_TRANSFORMERS
PRINT_ICFG_WITH_OUT_REGIONS
PRINT_ICFG_WITH_PRECONDITIONS
PRINT_ICFG_WITH_PROPER_EFFECTS
PRINT_ICFG_WITH_REGIONS
PRINT_ICFG_WITH_TRANSFORMERS

PRINT_INITIAL_PRECONDITION

PRINT_PARALLELIZED_CODE

PRINT_PARALLELIZED77_CODE
PRINT_PARALLELIZED90_CODE
PRINT_PARALLELIZEDCMF_CODE
PRINT_PARALLELIZEDCRAFT_CODE
PRINT_PARALLELIZEDCRAY_CODE
PRINT_PARALLELIZEDHPF_CODE
PRINT_PARALLELIZEDOMP_CODE

PRINT_SOURCE

PRINT_SOURCE
PRINT_SOURCE_COMPLEXITIES
PRINT_SOURCE_CONTINUATION_CONDITIONS
PRINT_SOURCE_CUMULATED_EFFECTS
PRINT_SOURCE_IN_EFFECTS
PRINT_SOURCE_IN_REGIONS
PRINT_SOURCE_OUT_EFFECTS
PRINT_SOURCE_OUT_REGIONS
PRINT_SOURCE_PRECONDITIONS
PRINT_SOURCE_PROPER_EFFECTS
PRINT_SOURCE_REGIONS

```
PRINT_SOURCE_TRANSFORMERS
```

6 Resources

The names of the current useful resources are given. These resources are computed by the `make` and `display` commands. These non-printable resources are encoded in internal data structures. The corresponding printable resources are listed in the following section. These resources and others are detailed the pipsmake documentation, `http://www.cri.ensmp.fr/pips/pipsmake-rc.htdoc` or `http://www.cri.ensmp.fr/pips/pipsmake-rc.htdoc/pipsmake-rc.pdf`.

```
CALLEES
CALLERS
CHAINS
COMPLEXITIES
DG
ENTITIES
SUMMARY_COMPLEXITY

## CODE

CODE
PARALLELIZED_CODE
PARSED_CODE


## EFFECTS

CUMULATED_EFFECTS
CUMULATED_IN_EFFECTS
IN_EFFECTS
IN_SUMMARY_EFFECTS
OUT_EFFECTS
OUT_SUMMARY_EFFECTS
PROPER_EFFECTS
SUMMARY_EFFECTS

## REGIONS

COPY_OUT_REGIONS
CUMULATED_IN_REGIONS
INV_IN_REGIONS
INV_REGIONS
IN_REGIONS
IN_SUMMARY_REGIONS
OUT_REGIONS
OUT_SUMMARY_REGIONS
PRIVATIZED_REGIONS
PROPER_REGIONS
```

```
REGIONS
SUMMARY_REGIONS

## REDUCTIONS

CUMULATED_REDUCTIONS
PROPER_REDUCTIONS
SUMMARY_REDUCTIONS

## REFERENCES

CUMULATED_REFERENCES
PROPER_REFERENCES

## PRECONDITION

INITIAL_PRECONDITION
PRECONDITIONS
PROGRAM_PRECONDITION
SUMMARY_PRECONDITION

## TRANSFORMERS

SUMMARY_TRANSFORMER
TRANSFORMERS

## CONTINUATION

MAY_CONTINUATION
MAY_SUMMARY_CONTINUATION
MUST_CONTINUATION
MUST_SUMMARY_CONTINUATION
```

7 Printable resources

The printable resources usually contain the string FILE. They can be computed by invoking the `display tpips` command. They are stored as human readable ASCII file in the workspace. The current resources which are printable follow, for more information see the pipsmake documentation, `http://www.cri.ensmp.fr/pips/pipsmake-rc.htdoc` or `http://www.cri.ensmp.fr/pips/pipsmake-rc.htdoc/pipsmake-rc.pdf`. .

```
CALLGRAPH_FILE                  # call graph file
DG_FILE                         # Dependence graph file
DVCG_FILE                       # Davinci Dependence graph file
FLINTED_FILE                    # source file with errors
GRAPH_PRINTED_FILE              # control graph
ICFG_FILE                       # interprocedural control flow graph
INITIAL_FILE                    # after splitting
PARALLELPRINTED_FILE            # parallel version
```

```
PRINTED_FILE                    # annoted sequential version
SOURCE_FILE                     # after the preprocessing phase
USER_FILE                       # after regeneration of user files
```

8 Options

Usage:
`tpips` [`-nscvh?jw`] [`-l` *logfile*] [`-r` *rcfile*] [`-e` *tpips-cmds*] *tpips-scripts*

-n no execution mode. Just check the script syntax;

-s behaves like a shell. `tpips` commands simply extend a shell;

-c behaves like a command, not a shell (it is the default option);

-v displays version and architecture information;

-h or -? provides some help;

-j jpips special mode;

-w starts with a wrapper (jpips special again)...

-l <logfile>: log to <logfile>;

-r <rcfile>: source the <rcfile> file (default `$HOME/.tpipsrc`);

-e tpips-cmds: executes the <tpips-cmds> commands.

9 Environment

Before using `tpips`, you need to add a PIPS root directory to your path and to set some PIPS environment variables. In order to do that, you can source from the PIPS distribution the shell script `pipsrc.sh` for any sh compatible shell, e.g. ksh or bash, or `pipsrc.csh` for any csh compatible shell, e.g. tcsh.

10 Small tools

`tpips` is the way to go with PIPS but it may be too complex to do some simple tasks used for some demonstrations or some well defined tasks.

10.1 `pips_c2openmp`

This tools parallelizes the C source files given in parameters and leaves the OpenMP generated files in the locally created database.
It is close to an automation of the example at section 4.6, but for C.

10.2 `pips_f2openmp`

This tools parallelizes the Fortran source files given in parameters and leaves the OpenMP generated files in the locally created database.
It is close to an automation of the example at section 4.6.