

OPARI 2

USER MANUAL

1.0.6 (revision 830)

Tue Oct 9 2012 02:17:57

Contents

1	Opari2	1
1.1	INSTALLATION	1
1.2	USAGE	2
1.3	CTC string decoding	4
1.4	LINKING (startup initialization only)	4
1.5	POMP user instrumentation	5
1.6	EXAMPLE	5
1.7	News	6
1.7.1	LINK STEP	6
1.7.2	POMP2	6
1.7.3	POMP2_Parallel_fork	6
1.7.4	pomp_tpd	7
1.7.5	Tasking construct	7
1.8	SUMMARY	8
Appendix A OPARI2 INSTALL		11
Appendix B Data Structure Documentation		19
B.1	POMP2_Region_info Struct Reference	19
B.1.1	Detailed Description	20
B.1.2	Field Documentation	20
Appendix C File Documentation		23
C.1	pomp2_lib.h File Reference	23
C.1.1	Detailed Description	24
C.1.2	Typedef Documentation	24
C.1.3	Function Documentation	24
C.2	pomp2_region_info.h File Reference	26
C.2.1	Detailed Description	26
C.2.2	Enumeration Type Documentation	27
C.2.3	Function Documentation	27

List of Figures

List of Tables

Chapter 1

Opari2

Opari2 is a tool to automatically instrument C, C++ and Fortran source code files in which OpenMP is used. Function calls to a [POMP2 API](#) are inserted around OpenMP directives. By implementing this API, detailed measurements regarding the runtime behavior of an OpenMP application can be made. A conforming POMP2 implementation needs to implement all POMP2 functions, see [pomp2_lib.h](#) for a list of those.

OpenMP 3.0 introduced tasking to OpenMP. To support this feature the POMP2 adapter needs to do some bookkeeping in regard to specific task IDs. The `pomp2_lib.c` provided with this package includes the necessary code so it is strongly advised to use it as a basis for writing an adapter to your own tool.

A detailed description of the first *Opari* version has been published by Mohr et al. in "Design and prototype of a performance tool interface for OpenMP" (Journal of supercomputing, 23, 2002).

1.1 INSTALLATION

Opari2 was developed with Autotools. After downloading and unpacking, change into your build directory and perform the following steps:

1. `./configure`
`[--prefix=<installation directory>]`
`[--with-compiler-suite=<gcc|ibm|intel|pathscale|pgi|studio>]`
2. `make`
3. `make install`

See the file `INSTALL` for further information.

1.2 USAGE

To create an instrumented version of an OpenMP application, each file of interest is transformed by the OPARI2 tool. The application is then linked against the POMP2 runtime measurement library and optionally to a special initialization file (see section [LINKING \(startup initialization only\)](#) and [SUMMARY](#) for further details).

A call to `Opari2` has the following syntax:

Usage: `opari2 [OPTION] ... infile [outfile]`

with following options and parameters:

<code>--f77 --f90 --c --c++</code>	[OPTIONAL] Specifies the programming language of the input source file. This option is only necessary if the automatic language detection based on the input file suffix fails.
<code>--nosrc</code>	[OPTIONAL] If specified, OPARI2 does not generate <code>#line</code> constructs, which allow to preserve the original source file and line number information, in the transformation process. This option might be necessary if the OpenMP compiler does not understand <code>#line</code> constructs. The default is to generate <code>#line</code> constructs.
<code>--nodecl</code>	[OPTIONAL] Disables the generation of <code>POMP2_DLISTXXXXX</code> macros. These are used in the parallel directives of the instrumentation to make the region handles shared. By using this option the shared clause is used directly on the parallel directive with the respective region handles.
<code>--tpd</code>	[OPTIONAL] Adds the clause <code>'copyin(<pomp_tpd>')</code> to any parallel construct. This allows to pass data from the creating thread to its children. The variable is declared externally in all files, so it needs to be defined by the pomp library.
<code>--disable=<constructs></code>	[OPTIONAL] Disable the instrumentation of manually-annotated POMP regions or the more fine-grained OpenMP constructs such as <code>!\$OMP ATOMIC</code> . <code><constructs></code> is a comma separated list of the constructs for which the instrumentation should be disabled. Accepted tokens are <code>atomic</code> , <code>critical</code> , <code>master</code> , <code>flush</code> , <code>single</code> , <code>ordered</code> or <code>locks</code> (as well as <code>sync</code> to disable all of them) or <code>regions</code> .
<code>--task=abort warn remove</code>	Special treatment for the task directive abort: Stop instrumentation with an error message when encountering a task directive. warn: Resume but print a warning. remove: Remove all task directives.
<code>--untied=abort keep no-warn</code>	Special treatment for the untied task attribute. The default behavior is to remove the untied

1.2 USAGE

	attribute, thus making all tasks tied, and print out a warning.
	abort: Stop instrumentation with an error message when encountering a task directive with the untied attribute.
	keep: Do not remove the untied attribute.
	no-warn: Do not print out a warning.
 [--tpd-mangling= gnu intel sun pgi ibm cray]	 [OPTIONAL] If programming languages are mixed (C and Fortran), the <pomp_tpd> needs to use the Fortran mangled name also in C files. This option specifies to use the mangling scheme of the gnu, intel, sun, pgi or ibm compiler. The default is to use the mangling scheme of the compiler used to build opari2.
 [--version]	 [OPTIONAL] Prints version information.
 [--help]	 [OPTIONAL] Prints this help text.
 infile	 Input file name.
 [outfile]	 [OPTIONAL] Output file name. If not specified, opari2 uses the name infile.mod.suffix if the input file is called infile.suffix.

Report bugs to <scorep-bugs@groups.tu-dresden.de>.

If you run Opari2 on the input file `example.c` it will create two files:

- `example.mod.c` is the instrumented version of `example.c`, i.e. it contains the original code plus calls to the [POMP2 API](#) referencing handles to the OpenMP regions identified by Opari2.
- `example.c.opari.inc` contains the OpenMP region handle definitions accompanied with all the relevant data needed by the handles. This compile time context (CTC) information is encoded into a string for maximum portability. For each region, the tuple (region_handle, ctc_string) is passed to an initializing function ([POMP2_Assign_handle\(\)](#)). All calls to these initializing functions are gathered in a function named `POMP2_Init_reg_XXX_YY`, where `XXX_YY` is unique for each compilation unit.

At some point during the runtime of the instrumented application, the region handles need to be initialized using the information stored in the CTC string. This can be done in one of two ways:

- during *startup* of the measurement/POMP2 system, or
- during *runtime* when a region handle is accessed for the first time.

We *highly* recommend using the first option as it incurs much less runtime overhead than the second one (no locking, no lookup needed). In this case all `POMP2_Init_reg_XXX_YY` functions introduced by opari2 need to be called. See [LINKING \(startup initialization only\)](#) for further details. For runtime initialization the ctc string as argument to the relevant [POMP2 function calls](#) is provided as an argument.

1.3 CTC string decoding

As mentioned above, we pass ctc strings to different POMP2 functions. These functions need to parse the string in order to process the encoded information. With [POMP2_Region_info](#) and [ctcString2RegionInfo\(\)](#) the opari2 package provides means of doing this, see [pomp2_region_info.h](#).

The CTC string is a string in the format "length*key=value*key=value*[key=value]**, for example:

```
*82*regionType=parallel*sscl=xmpl.c:61:61*escl=xmpl.c:66:66*hasIf=1**
```

Mandatory keys are:

- *regionType* Type of the region (here parallel)
- *sscl* First line of the region (usually with full path to file)
- *escl* Last line of the region

Optional keys are

- *hasNumThreads* Set if a numThreads clause is used in the OpenMP directive
- *hasIf* Set if an if clause is used
- *hasOrdered* Set if an ordered clause is used
- *hasReduction* Set if a reduction clause is used
- *hasSchedule* Set if a schedule clause is used
- *hasCollapse* Set if a collapse clause is used

The optional values are set to 0 by default, i.e. the presence of the key denotes the presence of the respective clause.

You can use the function [ctcString2RegionInfo\(\)](#) to decode CTC strings. It can be found in [pomp2_region_info.c](#) and [pomp2_region_info.h](#), installed under `<opari-prefix>/share/opari2/devel`.

1.4 LINKING (startup initialization only)

For startup initialization all POMP2_Init_reg_XXX_YY functions that can be found in the object files and libraries of the application are called. This is done by creating an additional compilation unit that contains calls to following POMP2 functions:

- [POMP2_Init_region\(\)](#),
- [POMP2_Get_num_regions\(\)](#), and
- [POMP2_Get_opari2_version\(\)](#).

1.5 POMP user instrumentation

The resulting object file is linked to the application. During startup of the measurement system the only thing to be done is to call `POMP2_Init_region()` which then calls all `POMP2_Init_reg_XXX_YY` functions.

In order to create the additional compilation unit (for example `pomp2_init_file.c`) the following command sequence can be used:

```
% `opari2-config --nm` <objs_and_libs> | \
`opari2-config --egrep` -i "pomp2_init_reg" | \
`opari2-config --egrep` " [TN] " | \
`opari2-config --awk-cmd` -f \
`opari2-config --awk-script` > pomp2_init_file.c
```

Here, `<objs_and_libs>` denotes the entire set of object files and libraries that were instrumented by `opari2`.

Due to portability reasons `nm`, `egrep` and `awk` are not called directly but via the provided `opari2-config` tool.

1.5 POMP user instrumentation

For manual user instrumentation the following pragmas are provided.

C/C++:

```
#pragma pomp inst init
#pragma pomp inst begin(region_name)
#pragma pomp inst altend(region_name)
#pragma pomp inst end(region_name)
```

Fortran:

```
!$POMP INST INIT
!$POMP INST BEGIN(region_name)
!$POMP INST ALTEND(region_name)
!$POMP INST END(region_name)
```

Users can specify code regions, like functions for example, with `INST BEGIN` and `INST END`. If a region contains several exit points like `return/break/exit/...` all but the last need to be marked with `INST ALTEND` pragmas. The `INST INIT` pragma should be used for initialization in the beginning of main, if no other initialization method is used. See the [EXAMPLE](#) section for an example on how to use user instrumentation.

1.6 EXAMPLE

The directory `<prefix>/share/opari2/doc/example` contains the following files:

```
example.c
example.f
Makefile
```

The Makefile contains all required information for building the instrumented and uninstrumented binaries. It demonstrates the compilation and linking steps as described above.

Additional examples which illustrate the use of user instrumentation can be found in `<prefix>/share/opari2/doc/example_user_instrumentation`. The folder contains the following files:

```
example_user_instrumentation.c
example_user_instrumentation.f
Makefile
```

1.7 News

1.7.1 LINK STEP

Opari2 uses a new mechanism to link files. The main advantage is, that no `opari.rc` file is needed anymore. Libraries can now be preinstrumented and parallel builds are supported. To achieve this, the handles for parallel regions are instrumented using a `ctc_string`.

1.7.2 POMP2

The POMP2 interface is not compatible with the original POMP interface. All functions of the new API begin with `POMP2_`. The declaration prototypes can be found in [pomp2_lib.h](#).

1.7.3 POMP2_Parallel_fork

The `POMP2_Parallel_fork()` call has an additional argument to pass the requested number of threads to the POMP2 library. This allows the library to prepare data structures and allocate memory for the threads before they are created. The value passed to the library is determined as follows:

- If a `num_threads` clause is present, the expression inside this clause is evaluated into a local variable `pomp_num_threads`. This variable is afterwards passed in the call to `POMP2_Parallel_fork()` and in the `num_threads` clause itself.
- If no `num_threads` clause is present, `omp_get_max_threads()` is used to determine the requested value for the next parallel region. This value is stored in `pomp_num_threads` and passed to the `POMP2_Parallel_fork()` call.

In Fortran, instead of `omp_get_max_threads()`, a wrapper function `pomp_get_max_threads_XXX_X` is used. This function is needed to avoid multiple definitions of `omp_get_max_threads()` since we do not know whether it is defined in the user code or not. Removing all definitions in the user code would require much more Fortran parsing than is done with opari2, since function definitions cannot easily be distinguished from variable definitions.

1.7.4 `pomp_tpd`

If it is necessary for the POMP2 library to pass information from the master thread to its children, the option `--tpd` can be used. *Opari2* uses the `copyin` clause to pass a threadprivate variable `pomp_tpd` to the newly spawned threads at the beginning of a parallel region. This is a 64 bit integer variable, since Fortran does not allow pointers. However a pointer can be stored in this variable, passed to child threads with the `copyin` clause (in C/C++ or Fortran) and later on be cast back to a pointer in the `pomp` library.

To support mixed programming (C/Fortran) the variable name depends on the name mangling of the Fortran compiler. This means, for GNU, Sun, Intel and PGI C compilers the variable is called `pomp_tpd_` and for IBM it is called `pomp_tpd` in C. In Fortran it is of course always called `pomp_tpd`. The `--tpd-mangling` option can be used to change this. The variable is declared extern in all program units, so the `pomp` library contains the actual variable declaration of `pomp_tpd` as a 64 bit integer.

1.7.5 Tasking construct

In *OpenMP 3.0* the new tasking construct was introduced. All parts of a program are now implicitly executed as tasks and the user gets the possibility of creating tasks that can be scheduled for asynchronous execution. Furthermore these tasks can be interrupted at certain scheduling points and resumed later on (see the OpenMP API 3.0 for more detailed information).

Opari2 instruments functions `POMP2_Task_create_begin` and `POMP2_Task_create_end` to allow the recording of the task creation time. For the task execution time, the functions `POMP2_Task_begin` and `POMP2_Task_end` are instrumented in the code. To correctly record a profile or a trace of a program execution these different instances of tasks need to be differentiated. Since OpenMP does not provide Task ids, the performance measurement system needs to create and maintain own task ids. This cannot be done by code instrumentation as done by *Opari2* alone but requires some administration of task ids during runtime. To allow the measurement system to administrate these ids, additional task id parameters (`pomp_old_task/pomp_new_task`) were added to all functions belonging to OpenMP constructs which are task scheduling points. With this package there is a "dummy" library, which can be used as an adapter to your measurement system. This library contains all the relevant functionality to keep track of the different instances of tasks and it is highly recommended to use it as a template to implement your own adapter for your measurement system.

For more detailed information on this mechanism see:

"How to Reconcile Event-Based Performance Analysis with Tasking in OpenMP"

by Daniel Lorenz, Bernd Mohr, Christian Rössel, Dirk Schmidl, and Felix Wolf

In: Proc. of 6th Int. Workshop of OpenMP (IWOMP), LNCS, vol. 6132, pp. 109121

DOI: 10.1007/978-3-642-13217-9_9

1.8 SUMMARY

The typical usage of OPARI2 consists of the following steps:

1. Call OPARI2 for each input source file

```
% opari2 file1.f90
...
% opari2 fileN.f90
```

2. Compile all modified output files *.mod.* using the OpenMP compiler

3. Generate the initialization file

```
% `opari2-config --nm` file1.mod.o ... fileN.mod.o | \
`opari2-config --egrep` -i "pomp2_init_reg" | \
`opari2-config --egrep` " [TD] " | \
`opari2-config --awk-cmd` -f \
`opari2-config --awk-script` > pomp2_init_file.c
```

4. Link the resulting object files against the pomp2 runtime measurement library.

Appendices

Appendix A

OPARI2 INSTALL

For generic installation instructions see below.

Configuration of OPARI2

Optional Features:

--disable-FEATURE	do not include FEATURE (same as --enable-FEATURE=no)
--enable-FEATURE[=ARG]	include FEATURE [ARG=yes]
--enable-silent-rules	less verbose build output (undo: 'make V=1')
--disable-silent-rules	verbose build output (undo: 'make V=0')
--disable-libtool-lock	avoid locking (might break parallel builds)
--disable-openmp	do not use OpenMP
--disable-option-checking	ignore unrecognized --enable/--with options
--disable-dependency-tracking	speeds up one-time build
--enable-dependency-tracking	do not reject slow dependency extractors
--enable-shared[=PKGS]	build shared libraries [default=no]
--enable-static[=PKGS]	build static libraries [default=yes]
--enable-fast-install[=PKGS]	optimize for fast installation [default=yes]

Optional Packages:

--with-PACKAGE[=ARG]	use PACKAGE [ARG=yes]
--without-PACKAGE	do not use PACKAGE (same as --with-PACKAGE=no)
--with-platform=(auto,disabled,<platform>)	autodetect platform [auto], disabled or select one from: altix, aix, arm, bgl, bgp, bgq, crayxt, linux, solaris, mac, necsx.
--with-compiler-suite=(gcc ibm intel pathscale pgi studio)	The compiler suite to build this package with. Needs to be in \$PATH [gcc].
--with-pic	try to use only PIC/non-PIC objects [default=use both]
--with-gnu-ld	assume the C compiler uses GNU ld [default=no]
--with-sysroot=DIR	Search for dependent libraries within DIR (or the compiler's sysroot if not specified).

Some influential environment variables:

(note that the `_FOR_BUILD` variables take precedence, e.g. if you call `opari's configure` from a top level `configure` in a cross-compile environment that defines `CC` as well as `CC_FOR_BUILD` etc.)

`CC_FOR_BUILD`

C compiler command for the frontend build

`CXX_FOR_BUILD`

APPENDIX A. OPARI2 INSTALL

C++ compiler command for the frontend build
F77_FOR_BUILD Fortran 77 compiler command for the frontend build
FC_FOR_BUILD Fortran compiler command for the frontend build
CPPFLAGS_FOR_BUILD (Objective) C/C++ preprocessor flags for the frontend build, e.g. -I<include dir> if you have headers in a nonstandard directory <include dir>
CFLAGS_FOR_BUILD C compiler flags for the frontend build
CXXFLAGS_FOR_BUILD C++ compiler flags for the frontend build
FFLAGS_FOR_BUILD Fortran 77 compiler flags for the frontend build
FCFLAGS_FOR_BUILD Fortran compiler flags for the frontend build
LDFLAGS_FOR_BUILD linker flags for the frontend build, e.g. -L<lib dir> if you have libraries in a nonstandard directory <lib dir>
LIBS_FOR_BUILD libraries to pass to the linker for the frontend build, e.g. -l<library>
CC C compiler command
CFLAGS C compiler flags
LDFLAGS linker flags, e.g. -L<lib dir> if you have libraries in a nonstandard directory <lib dir>
LIBS libraries to pass to the linker, e.g. -l<library>
CPPFLAGS (Objective) C/C++ preprocessor flags, e.g. -I<include dir> if you have headers in a nonstandard directory <include dir>
CXX C++ compiler command
CXXFLAGS C++ compiler flags
F77 Fortran 77 compiler command
FFLAGS Fortran 77 compiler flags
FC Fortran compiler command
FCFLAGS Fortran compiler flags
CPP C preprocessor
CXXCPP C++ preprocessor

Use these variables to override the choices made by 'configure' or to help it to find libraries and programs with nonstandard names/locations.

Please report bugs to <scorep-bugs@groups.tu-dresden.de>.

Installation Instructions

Copyright (C) 1994, 1995, 1996, 1999, 2000, 2001, 2002, 2004, 2005, 2006, 2007, 2008, 2009 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification, are permitted in any medium without royalty provided the copyright notice and this notice are preserved. This file is offered as-is, without warranty of any kind.

Basic Installation
=====

Briefly, the shell commands './configure; make; make install' should configure, build, and install this package. The following more-detailed instructions are generic; see the 'README' file for

instructions specific to this package. Some packages provide this 'INSTALL' file but do not implement all of the features documented below. The lack of an optional feature in a given package is not necessarily a bug. More recommendations for GNU packages can be found in **note Makefile Conventions: (standards)Makefile Conventions*.

The 'configure' shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a 'Makefile' in each directory of the package. It may also create one or more '.h' files containing system-dependent definitions. Finally, it creates a shell script 'config.status' that you can run in the future to recreate the current configuration, and a file 'config.log' containing compiler output (useful mainly for debugging 'configure').

It can also use an optional file (typically called 'config.cache' and enabled with '--cache-file=config.cache' or simply '-C') that saves the results of its tests to speed up reconfiguring. Caching is disabled by default to prevent problems with accidental use of stale cache files.

If you need to do unusual things to compile the package, please try to figure out how 'configure' could check whether to do them, and mail diffs or instructions to the address given in the 'README' so they can be considered for the next release. If you are using the cache, and at some point 'config.cache' contains results you don't want to keep, you may remove or edit it.

The file 'configure.ac' (or 'configure.in') is used to create 'configure' by a program called 'autoconf'. You need 'configure.ac' if you want to change it or regenerate 'configure' using a newer version of 'autoconf'.

The simplest way to compile this package is:

1. 'cd' to the directory containing the package's source code and type './configure' to configure the package for your system.

Running 'configure' might take a while. While running, it prints some messages telling which features it is checking for.

2. Type 'make' to compile the package.
3. Optionally, type 'make check' to run any self-tests that come with the package, generally using the just-built uninstalled binaries.
4. Type 'make install' to install the programs and any data files and documentation. When installing into a prefix owned by root, it is recommended that the package be configured and built as a regular user, and only the 'make install' phase executed with root privileges.
5. Optionally, type 'make installcheck' to repeat any self-tests, but this time using the binaries in their final installed location. This target does not install anything. Running this target as a regular user, particularly if the prior 'make install' required root privileges, verifies that the installation completed correctly.
6. You can remove the program binaries and object files from the source code directory by typing 'make clean'. To also remove the files that 'configure' created (so you can compile the package for

a different kind of computer), type 'make distclean'. There is also a 'make maintainer-clean' target, but that is intended mainly for the package's developers. If you use it, you may have to get all sorts of other programs in order to regenerate files that came with the distribution.

7. Often, you can also type 'make uninstall' to remove the installed files again. In practice, not all packages have tested that uninstallation works correctly, even though it is required by the GNU Coding Standards.
8. Some packages, particularly those that use Automake, provide 'make distcheck', which can be used by developers to test that all other targets like 'make install' and 'make uninstall' work correctly. This target is generally not run by end users.

Compilers and Options

=====

Some systems require unusual options for compilation or linking that the 'configure' script does not know about. Run './configure --help' for details on some of the pertinent environment variables.

You can give 'configure' initial values for configuration parameters by setting variables in the command line or in the environment. Here is an example:

```
./configure CC=c99 CFLAGS=-g LIBS=-lposix
```

*Note Defining Variables::, for more details.

Compiling For Multiple Architectures

=====

You can compile the package for more than one kind of computer at the same time, by placing the object files for each architecture in their own directory. To do this, you can use GNU 'make'. 'cd' to the directory where you want the object files and executables to go and run the 'configure' script. 'configure' automatically checks for the source code in the directory that 'configure' is in and in '..'. This is known as a "VPATH" build.

With a non-GNU 'make', it is safer to compile the package for one architecture at a time in the source code directory. After you have installed the package for one architecture, use 'make distclean' before reconfiguring for another architecture.

On MacOS X 10.5 and later systems, you can create libraries and executables that work on multiple system types--known as "fat" or "universal" binaries--by specifying multiple '-arch' options to the compiler but only a single '-arch' option to the preprocessor. Like this:

```
./configure CC="gcc -arch i386 -arch x86_64 -arch ppc -arch ppc64" \  
CXX="g++ -arch i386 -arch x86_64 -arch ppc -arch ppc64" \  
CPP="gcc -E" CXXCPP="g++ -E"
```

This is not guaranteed to produce working output in all cases, you may have to build one architecture at a time and combine the results using the 'lipo' tool if you have problems.

Installation Names

=====

By default, 'make install' installs the package's commands under '/usr/local/bin', include files under '/usr/local/include', etc. You can specify an installation prefix other than '/usr/local' by giving 'configure' the option '--prefix=PREFIX', where PREFIX must be an absolute file name.

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you pass the option '--exec-prefix=PREFIX' to 'configure', the package uses PREFIX as the prefix for installing programs and libraries. Documentation and other data files still use the regular prefix.

In addition, if you use an unusual directory layout you can give options like '--bindir=DIR' to specify different values for particular kinds of files. Run 'configure --help' for a list of the directories you can set and what kinds of files go in them. In general, the default for these options is expressed in terms of '\${prefix}', so that specifying just '--prefix' will affect all of the other directory specifications that were not explicitly provided.

The most portable way to affect installation locations is to pass the correct locations to 'configure'; however, many packages provide one or both of the following shortcuts of passing variable assignments to the 'make install' command line to change installation locations without having to reconfigure or recompile.

The first method involves providing an override variable for each affected directory. For example, 'make install prefix=/alternate/directory' will choose an alternate location for all directory configuration variables that were expressed in terms of '\${prefix}'. Any directories that were specified during 'configure', but not in terms of '\${prefix}', must each be overridden at install time for the entire installation to be relocated. The approach of makefile variable overrides for each directory variable is required by the GNU Coding Standards, and ideally causes no recompilation. However, some platforms have known limitations with the semantics of shared libraries that end up requiring recompilation when using this method, particularly noticeable in packages that use GNU Libtool.

The second method involves providing the 'DESTDIR' variable. For example, 'make install DESTDIR=/alternate/directory' will prepend '/alternate/directory' before all installation names. The approach of 'DESTDIR' overrides is not required by the GNU Coding Standards, and does not work on platforms that have drive letters. On the other hand, it does better at avoiding recompilation issues, and works well even when some directory options were not specified in terms of '\${prefix}' at 'configure' time.

Optional Features

=====

If the package supports it, you can cause programs to be installed with an extra prefix or suffix on their names by giving 'configure' the option '--program-prefix=PREFIX' or '--program-suffix=SUFFIX'.

Some packages pay attention to '--enable-FEATURE' options to 'configure', where FEATURE indicates an optional part of the package. They may also pay attention to '--with-PACKAGE' options, where PACKAGE is something like 'gnu-as' or 'x' (for the X Window System). The 'README' should mention any '--enable-' and '--with-' options that the

package recognizes.

For packages that use the X Window System, 'configure' can usually find the X include and library files automatically, but if it doesn't, you can use the 'configure' options '--x-includes=DIR' and '--x-libraries=DIR' to specify their locations.

Some packages offer the ability to configure how verbose the execution of 'make' will be. For these packages, running './configure --enable-silent-rules' sets the default to minimal output, which can be overridden with 'make V=1'; while running './configure --disable-silent-rules' sets the default to verbose, which can be overridden with 'make V=0'.

Particular systems
=====

On HP-UX, the default C compiler is not ANSI C compatible. If GNU CC is not installed, it is recommended to use the following options in order to use an ANSI C compiler:

```
./configure CC="cc -Ae -D_XOPEN_SOURCE=500"
```

and if that doesn't work, install pre-built binaries of GCC for HP-UX.

On OSF/1 a.k.a. Tru64, some versions of the default C compiler cannot parse its '<wchar.h>' header file. The option '-nodtk' can be used as a workaround. If GNU CC is not installed, it is therefore recommended to try

```
./configure CC="cc"
```

and if that doesn't work, try

```
./configure CC="cc -nodtk"
```

On Solaris, don't put '/usr/ucb' early in your 'PATH'. This directory contains several dysfunctional programs; working variants of these programs are available in '/usr/bin'. So, if you need '/usr/ucb' in your 'PATH', put it after '/usr/bin'.

On Haiku, software installed for all users goes in '/boot/common', not '/usr/local'. It is recommended to use the following options:

```
./configure --prefix=/boot/common
```

Specifying the System Type
=====

There may be some features 'configure' cannot figure out automatically, but needs to determine by the type of machine the package will run on. Usually, assuming the package is built to be run on the same architectures, 'configure' can figure that out, but if it prints a message saying it cannot guess the machine type, give it the '--build=TYPE' option. TYPE can either be a short name for the system type, such as 'sun4', or a canonical name which has the form:

```
CPU-COMPANY-SYSTEM
```

where SYSTEM can have one of these forms:

```
OS
```

KERNEL-OS

See the file 'config.sub' for the possible values of each field. If 'config.sub' isn't included in this package, then this package doesn't need to know the machine type.

If you are building compiler tools for cross-compiling, you should use the option '--target=TYPE' to select the type of system they will produce code for.

If you want to use a cross compiler, that generates code for a platform different from the build platform, you should specify the "host" platform (i.e., that on which the generated programs will eventually be run) with '--host=TYPE'.

Sharing Defaults

=====

If you want to set default values for 'configure' scripts to share, you can create a site shell script called 'config.site' that gives default values for variables like 'CC', 'cache_file', and 'prefix'. 'configure' looks for 'PREFIX/share/config.site' if it exists, then 'PREFIX/etc/config.site' if it exists. Or, you can set the 'CONFIG_SITE' environment variable to the location of the site script. A warning: not all 'configure' scripts look for a site script.

Defining Variables

=====

Variables not defined in a site shell script can be set in the environment passed to 'configure'. However, some packages may run configure again during the build, and the customized values of these variables may be lost. In order to avoid this problem, you should set them in the 'configure' command line, using 'VAR=value'. For example:

```
./configure CC=/usr/local2/bin/gcc
```

causes the specified 'gcc' to be used as the C compiler (unless it is overridden in the site shell script).

Unfortunately, this technique does not work for 'CONFIG_SHELL' due to an Autoconf bug. Until the bug is fixed you can use this workaround:

```
CONFIG_SHELL=/bin/bash /bin/bash ./configure CONFIG_SHELL=/bin/bash
```

'configure' Invocation

=====

'configure' recognizes the following options to control how it operates.

'--help'

'-h'

Print a summary of all of the options to 'configure', and exit.

'--help=short'

'--help=recursive'

Print a summary of the options unique to this package's 'configure', and exit. The 'short' variant lists options used only in the top level, while the 'recursive' variant lists options also present in any nested packages.

```
'--version'
'-v'
    Print the version of Autoconf used to generate the 'configure'
    script, and exit.

'--cache-file=FILE'
    Enable the cache: use and save the results of the tests in FILE,
    traditionally 'config.cache'. FILE defaults to '/dev/null' to
    disable caching.

'--config-cache'
'-C'
    Alias for '--cache-file=config.cache'.

'--quiet'
'--silent'
'-q'
    Do not print messages saying which checks are being made. To
    suppress all normal output, redirect it to '/dev/null' (any error
    messages will still be shown).

'--srcdir=DIR'
    Look for the package's source code in directory DIR. Usually
    'configure' can determine that directory automatically.

'--prefix=DIR'
    Use DIR as the installation prefix. *note Installation Names::
    for more details, including other options available for fine-tuning
    the installation locations.

'--no-create'
'-n'
    Run the configure checks, but stop before creating any output
    files.

'configure' also accepts some other, not widely useful, options. Run
'configure --help' for more details.
```


Appendix B

Data Structure Documentation

B.1 POMP2_Region_info Struct Reference

This struct stores all information on an OpenMP region, like the region type or corresponding source lines. The function [ctcString2RegionInfo\(\)](#) can be used to fill this struct with data from a ctcString.

```
#include <pomp2_region_info.h>
```

Data Fields

Required attributes

- [POMP2_Region_type](#) mRegionType
- char * [mStartFileName](#)
- unsigned [mStartLine1](#)
- unsigned [mStartLine2](#)
- char * [mEndFileName](#)
- unsigned [mEndLine1](#)
- unsigned [mEndLine2](#)

Currently not provided by opari

- bool [mHasCopyIn](#)
- bool [mHasCopyPrivate](#)
- bool [mHasIf](#)
- bool [mHasFirstPrivate](#)
- bool [mHasLastPrivate](#)
- bool [mHasNoWait](#)
- bool [mHasNumThreads](#)
- bool [mHasOrdered](#)
- bool [mHasReduction](#)
- bool [mHasCollapse](#)
- bool [mHasUntied](#)
- [POMP2_Schedule_type](#) mScheduleType

- char * [mUserGroupName](#)

Attributes for specific region types

- unsigned [mNumSections](#)
- char * [mCriticalName](#)
- char * [mUserRegionName](#)

B.1.1 Detailed Description

This struct stores all information on an OpenMP region, like the region type or corresponding source lines. The function [ctcString2RegionInfo\(\)](#) can be used to fill this struct with data from a ctcString.

B.1.2 Field Documentation

B.1.2.1 char* POMP2_Region_info::mCriticalName

name of a named critical region

B.1.2.2 char* POMP2_Region_info::mEndFileName

name of the corresponding source file from the closing pragma

B.1.2.3 unsigned POMP2_Region_info::mEndLine1

line number of the first line from the closing pragma

B.1.2.4 unsigned POMP2_Region_info::mEndLine2

line number of the last line from the closing pragma

B.1.2.5 bool POMP2_Region_info::mHasCollapse

true if a collapse clause is present

B.1.2.6 bool POMP2_Region_info::mHasCopyIn

true if a copyin clause is present

B.1.2.7 bool POMP2_Region_info::mHasCopyPrivate

true if a copyprivate clause is present

B.1 POMP2_Region_info Struct Reference

B.1.2.8 bool POMP2_Region_info::mHasFirstPrivate

true if a firstprivate clause is present

B.1.2.9 bool POMP2_Region_info::mHasIf

true if an if clause is present

B.1.2.10 bool POMP2_Region_info::mHasLastPrivate

true if a lastprivate clause is present

B.1.2.11 bool POMP2_Region_info::mHasNoWait

true if a nowait clause is present

B.1.2.12 bool POMP2_Region_info::mHasNumThreads

true if a numThreads clause is present

B.1.2.13 bool POMP2_Region_info::mHasOrdered

true if an ordered clause is present

B.1.2.14 bool POMP2_Region_info::mHasReduction

true if a reduction clause is present

B.1.2.15 bool POMP2_Region_info::mHasUntied

true if a untied clause was present, even if the task was changed to tied during instrumentation.

B.1.2.16 unsigned POMP2_Region_info::mNumSections

number of sections

B.1.2.17 POMP2_Region_type POMP2_Region_info::mRegionType

type of the OpenMP region

APPENDIX B. DATA STRUCTURE DOCUMENTATION

B.1.2.18 `POMP2_Schedule_type` `POMP2_Region_info::mScheduleType`

schedule type in the schedule clause

B.1.2.19 `char*` `POMP2_Region_info::mStartFileName`

name of the corresponding source file from the opening pragma

B.1.2.20 `unsigned` `POMP2_Region_info::mStartLine1`

line number of the first line from the opening pragma

B.1.2.21 `unsigned` `POMP2_Region_info::mStartLine2`

line number of the last line from the opening pragma

B.1.2.22 `char*` `POMP2_Region_info::mUserGroupName`

user group name

B.1.2.23 `char*` `POMP2_Region_info::mUserRegionName`

name of a user defined region

The documentation for this struct was generated from the following file:

- [pomp2_region_info.h](#)

Appendix C

File Documentation

C.1 pomp2.lib.h File Reference

This file contains the declarations of all POMP2 functions.

```
#include <stddef.h>
```

```
#include <stdint.h>
```

Typedefs

- typedef void * [POMP2_Region_handle](#)

Functions

- void [POMP2_Assign_handle](#) ([POMP2_Region_handle](#) *pomp2_handle, const char ctc_string[])
- void [POMP2_Begin](#) ([POMP2_Region_handle](#) *pomp2_handle)
- void [POMP2_End](#) ([POMP2_Region_handle](#) *pomp2_handle)
- void [POMP2_Finalize](#) ()
- [POMP2_Task_handle](#) [POMP2_Get_new_task_handle](#) ()
- void [POMP2_Init](#) ()
- void [POMP2_Off](#) ()
- void [POMP2_On](#) ()

Functions generated by the instrumenter

- size_t [POMP2_Get_num_regions](#) ()
- void [POMP2_Init_regions](#) ()
- const char * [POMP2_Get_opari2_version](#) ()

C.1.1 Detailed Description

This file contains the declarations of all POMP2 functions. alpha

Authors

Daniel Lorenz <d.lorenz@fz-juelich.de> Dirk Schmidl <schmidl@rz.rwth-aachen.de>
Peter Philippen <p.philippen@fz-juelich.de>

C.1.2 Typedef Documentation

C.1.2.1 typedef void* POMP2_Region_handle

Handles to identify OpenMP regions.

C.1.3 Function Documentation

C.1.3.1 void POMP2_Assign_handle (POMP2_Region_handle * *pomp2_handle*, const char *ctc_string*[])

Registers a POMP2 region and returns a region handle.

Parameters

<i>pomp2_-handle</i>	Returns the handle for the newly registered region.
<i>ctc_string</i>	A string containing the region data.

C.1.3.2 void POMP2_Begin (POMP2_Region_handle * *pomp2_handle*)

Called at the begin of a user defined POMP2 region.

Parameters

<i>pomp2_-handle</i>	The handle of the started region.
----------------------	-----------------------------------

C.1.3.3 void POMP2_End (POMP2_Region_handle * *pomp2_handle*)

Called at the begin of a user defined POMP2 region.

Parameters

<i>pomp2_-handle</i>	The handle of the started region.
----------------------	-----------------------------------

C.1 pomp2_lib.h File Reference

C.1.3.4 void POMP2_Finalize ()

Finalizes the POMP2 adapter. It is inserted at the #pragma pomp inst end.

C.1.3.5 POMP2.Task.handle POMP2_Get_new_task_handle ()

Function that returns a new task handle.

Returns

new task handle

C.1.3.6 size_t POMP2_Get_num_regions ()

Returns the number of instrumented regions.

The instrumenter scans all opari-created include files with nm and greps the POMP2_INIT_uuid_numRegions() function calls. Here we return the sum of all numRegions.

Returns

number of instrumented regions

C.1.3.7 const char* POMP2_Get_opari2_version ()

Returns the opari version.

Returns

version string

C.1.3.8 void POMP2_Init ()

Initializes the POMP2 adapter. It is inserted at the #pragma pomp inst begin.

C.1.3.9 void POMP2_Init_regions ()

Init all opari-created regions.

The instrumentor scans all opari-created include files with nm and greps the POMP2_INIT_uuid_numRegions() function calls. The instrumentor then defines these functions by calling all grepped functions.

C.1.3.10 void POMP2_Off ()

Disables the POMP2 adapter.

C.1.3.11 void POMP2_On ()

Enables the POMP2 adapter.

C.2 pomp2_region_info.h File Reference

This file contains function declarations and structs which handle informations on OpenMP regions. [POMP2_Region_info](#) is used to store these informations. It can be filled with a ctcString by [ctcString2RegionInfo\(\)](#).

```
#include <stdbool.h>
```

Data Structures

- struct [POMP2_Region_info](#)

This struct stores all information on an OpenMP region, like the region type or corresponding source lines. The function [ctcString2RegionInfo\(\)](#) can be used to fill this struct with data from a ctcString.

Enumerations

- enum [POMP2_Region_type](#)
- enum [POMP2_Schedule_type](#)

Functions

- void [ctcString2RegionInfo](#) (const char ctcString[], [POMP2_Region_info](#) *regionInfo)
- void [freePOMP2RegionInfoMembers](#) ([POMP2_Region_info](#) *regionInfo)
- const char * [pomp2RegionType2String](#) ([POMP2_Region_type](#) regionType)
- const char * [pomp2ScheduleType2String](#) ([POMP2_Schedule_type](#) scheduleType)

C.2.1 Detailed Description

This file contains function declarations and structs which handle informations on OpenMP regions. [POMP2_Region_info](#) is used to store these informations. It can be filled with a ctcString by [ctcString2RegionInfo\(\)](#).

Author

Christian Rössel <c.roessel@fz-juelich.de> alpha

Date

Started Fri Mar 20 16:30:45 2009

C.2 pomp2_region_info.h File Reference

C.2.2 Enumeration Type Documentation

C.2.2.1 enum POMP2_Region_type

POMP2_Region_type

C.2.2.2 enum POMP2_Schedule_type

type to store the scheduling type of a for worksharing construct

C.2.3 Function Documentation

C.2.3.1 void ctcString2RegionInfo (const char *ctcString*[], POMP2_Region_info * *regionInfo*)

[ctcString2RegionInfo\(\)](#) fills the [POMP2_Region_info](#) object with data read from the *ctcString*. If the *ctcString* does not comply with the specification, the program aborts with exit code 1.

Rationale: [ctcString2RegionInfo\(\)](#) is used during initialization of the measurement system. If an error occurs, it is better to abort than to struggle with undefined behaviour or *guessing* the meaning of the broken string.

Note

Can be called from multiple threads concurrently, assuming malloc is thread-safe. [ctcString2RegionInfo\(\)](#) will assign memory to the members of *regionInfo*. You are supposed to release this memory by calling [freePOMP2RegionInfoMembers\(\)](#).

Parameters

<i>ctcString</i>	A string in the format "length*key=value*[key=value]*". The length field is parsed but not used by this implementation. Possible values for key are listed in ctcTokenMap . The string must at least contain values for the keys <i>regionType</i> , <i>sscl</i> and <i>escl</i> . Possible values for the key <i>regionType</i> are listed in regionTypesMap . The format for <i>sscl</i> resp. <i>escl</i> values is "filename:lineNo1:lineNo2".
<i>regionInfo</i>	must be a valid object

Postcondition

At least the required attributes (see [POMP2_Region_info](#)) are set.

All other members of *regionInfo* are set to 0 resp. false resp. POMP2_No_schedule.

If *regionType*=sections than [POMP2_Region_info::mNumSections](#) has a value > 0.

If *regionType*=region than [POMP2_Region_info::mUserRegionName](#) has a value != 0.

If *regionType*=critical than [POMP2_Region_info::mCriticalName](#) may have a value != 0.

C.2.3.2 void freePOMP2RegionInfoMembers (POMP2_Region_info * *regionInfo*)

Free the memory of the regionInfo members.

Parameters

<i>regionInfo</i>	The regioninfo to be freed.
-------------------	-----------------------------

C.2.3.3 const char* pomp2RegionType2String (POMP2_Region_type *regionType*)

converts regionType into a string

Parameters

<i>regionType</i>	The regionType to be converted.
-------------------	---------------------------------

Returns

string representation of the region type

C.2.3.4 const char* pomp2ScheduleType2String (POMP2_Schedule_type *scheduleType*)

converts scheduleType into a string

Parameters

<i>schedule- Type</i>	The scheduleType to be converted.
---------------------------	-----------------------------------

Returns

string representation of the scheduleType

Index

ctcString2RegionInfo
 pomp2_region_info.h, [27](#)

freePOMP2RegionInfoMembers
 pomp2_region_info.h, [28](#)

mCriticalName
 POMP2_Region_info, [20](#)

mEndFileName
 POMP2_Region_info, [20](#)

mEndLine1
 POMP2_Region_info, [20](#)

mEndLine2
 POMP2_Region_info, [20](#)

mHasCollapse
 POMP2_Region_info, [20](#)

mHasCopyIn
 POMP2_Region_info, [20](#)

mHasCopyPrivate
 POMP2_Region_info, [20](#)

mHasFirstPrivate
 POMP2_Region_info, [20](#)

mHasIf
 POMP2_Region_info, [21](#)

mHasLastPrivate
 POMP2_Region_info, [21](#)

mHasNoWait
 POMP2_Region_info, [21](#)

mHasNumThreads
 POMP2_Region_info, [21](#)

mHasOrdered
 POMP2_Region_info, [21](#)

mHasReduction
 POMP2_Region_info, [21](#)

mHasUntied
 POMP2_Region_info, [21](#)

mNumSections
 POMP2_Region_info, [21](#)

mRegionType
 POMP2_Region_info, [21](#)

mScheduleType
 POMP2_Region_info, [21](#)

mStartFileName
 POMP2_Region_info, [22](#)

mStartLine1
 POMP2_Region_info, [22](#)

mStartLine2
 POMP2_Region_info, [22](#)

mUserGroupName
 POMP2_Region_info, [22](#)

mUserRegionName
 POMP2_Region_info, [22](#)

POMP2_Assign_handle
 pomp2_lib.h, [24](#)

POMP2_Begin
 pomp2_lib.h, [24](#)

POMP2_End
 pomp2_lib.h, [24](#)

POMP2_Finalize
 pomp2_lib.h, [24](#)

POMP2_Get_new_task_handle
 pomp2_lib.h, [25](#)

POMP2_Get_num_regions
 pomp2_lib.h, [25](#)

POMP2_Get_opari2_version
 pomp2_lib.h, [25](#)

POMP2_Init
 pomp2_lib.h, [25](#)

POMP2_Init_regions
 pomp2_lib.h, [25](#)

pomp2_lib.h, [23](#)
 POMP2_Assign_handle, [24](#)
 POMP2_Begin, [24](#)
 POMP2_End, [24](#)
 POMP2_Finalize, [24](#)
 POMP2_Get_new_task_handle, [25](#)
 POMP2_Get_num_regions, [25](#)
 POMP2_Get_opari2_version, [25](#)
 POMP2_Init, [25](#)
 POMP2_Init_regions, [25](#)
 POMP2_Off, [25](#)

- POMP2_On, [25](#)
- POMP2_Region_handle, [24](#)
- POMP2_Off
 - pomp2_lib.h, [25](#)
- POMP2_On
 - pomp2_lib.h, [25](#)
- POMP2_Region_handle
 - pomp2_lib.h, [24](#)
- POMP2_Region_info, [19](#)
 - mCriticalName, [20](#)
 - mEndFileName, [20](#)
 - mEndLine1, [20](#)
 - mEndLine2, [20](#)
 - mHasCollapse, [20](#)
 - mHasCopyIn, [20](#)
 - mHasCopyPrivate, [20](#)
 - mHasFirstPrivate, [20](#)
 - mHasIf, [21](#)
 - mHasLastPrivate, [21](#)
 - mHasNoWait, [21](#)
 - mHasNumThreads, [21](#)
 - mHasOrdered, [21](#)
 - mHasReduction, [21](#)
 - mHasUntied, [21](#)
 - mNumSections, [21](#)
 - mRegionType, [21](#)
 - mScheduleType, [21](#)
 - mStartFileName, [22](#)
 - mStartLine1, [22](#)
 - mStartLine2, [22](#)
 - mUserGroupName, [22](#)
 - mUserRegionName, [22](#)
- pomp2_region_info.h, [26](#)
 - ctcString2RegionInfo, [27](#)
 - freePOMP2RegionInfoMembers, [28](#)
 - POMP2_Region_type, [27](#)
 - POMP2_Schedule_type, [27](#)
 - pomp2RegionType2String, [28](#)
 - pomp2ScheduleType2String, [28](#)
- POMP2_Region_type
 - pomp2_region_info.h, [27](#)
- POMP2_Schedule_type
 - pomp2_region_info.h, [27](#)
- pomp2RegionType2String
 - pomp2_region_info.h, [28](#)
- pomp2ScheduleType2String
 - pomp2_region_info.h, [28](#)