Portals4 GNU UPC User Manual

Contents

Chapter 1

Authors and Revision Information

Authors: Gary Funck <gary@intrepid.com> Nenad Vukicevic <nenad@intrepid.com>

Intrepid Technology, Inc.

http://www.intrepid.com

http://www.gccupc.org

Revision: 1.4 (2013/05/21)

Chapter 2

Introduction

The GNU UPC (GUPC) toolset provides a compilation and execution environment for programs written in the UPC (Unified Parallel C) language. The GNU UPC compiler extends the capabilities of the GNU GCC compiler.

The GNU UPC for Portals 4.0 (Portals4, Portals) is an implementation of GNU UPC that uses Portals interface for message passing between UPC threads running on separate nodes in a system area network.

Chapter 3

Requirements

3.1 Supported Systems

GUPC for Portals4 64-bit supported on the systems based on the Red Hat Linux (e.g. RHEL, Fedora, Scientific Linux, CentOS) and Infiniband networking support.

3.2 Additional Packages

To build the GUPC compiler, various special purpose libraries must be previously installed. The easiest method of installing these packages is to install them from binary packages located at the package repository associated with the particular OS that you are using; administrator privileges are required to install these packages. The list of packages needed is detailed here: http://www.gccupc.org/gnu-upc-info/gnu-upc-prerequisites

For example, on Redhat-based systems, the following packages must be installed: gmp-devel, mpfr-devel, libmpc-devel, and numactl-devel.

Some tips on installing those packages can be found under FAQ section on the gccupc website: http://www.gccupc.org/faq.html

The GCC pre-requisites page may also provide additional useful information: http://gcc.gnu.org/install/prerequisites.html

3.3 Portals4 Library

Portals 4 Reference Library Implementation must be installed on the system for GNU UPC to build and run.

See Portals4 Reference Implementation at http://code.google.com/p/portals4/.

Chapter 4

Installing GNU UPC

Like most GNU software, GUPC must be configured before it can be built. This chapter describes the recommended configuration procedure with emphasis on the GUPC specific configuration options, as well as other common options.

More information on configuring GNU GCC can be found on the gcc.gnu.org website: http://gcc.gnu.org/install/configure.html

4.1 Configuration

We use *srcdir* to refer to the top-level source directory for GNU UPC; we use *objdir* to refer to the top-level build/object directory.

We highly recommend that GNU UPC be built into a separate directory from the sources which does not reside within the source tree. This is how generally GCC is also built.

When configuring GNU UPC, either cc or gcc must be in your path or you must set CC in your environment before running configure. Otherwise the configuration scripts may fail.

If you have previously built GNU UPC in the same directory, do 'make distclean' to delete all files that might be invalid. One of the files this deletes is Makefile; if 'make distclean' complains that Makefile does not exist or issues a message like "don't know how to make distclean" it probably means that the directory is already suitably clean.

To configure GNU UPC with Portals4 runtime support:

```
% mkdir objdir
% cd objdir
% srcdir/configure [options] --with-upc-runtime=portals4 \
  --enable-languages=c,upc
```

Sample configuration:

```
% srcdir/configure \
      --prefix=/usr/local/gupc \
      --with-upc-runtime=portals4 \
      --with-portals4=/usr/local/gupc-p4 \
      --enable-languages=c,upc
```

4.2 Build and Install

To build GNU UPC after the configuration step:

```
% make
% make install
```

An optional "-j" argument to the make command line can be used to improve the build time. On systems that have multiple cores, the "-j" can noticeably improve build times. As a general rule, set the value of "N" in "-jN" to about 1.5 times the number of available cores.

4.3 Configuration Options

The following GCC and GNU UPC options are provided to better tailor GNU UPC for your system. The full list of additional GCC configuration options can be found on the GCC web page http://gcc.gnu.org/install/configure.html

--prefix=*dirname*
Specify the top-level installation directory. This is the recommended method to install the tools into a directory other than the default. The top-level installation directory defaults to */usr/local*. For GNU UPC we recommend */usr/local/gupc*.

--enable-upc-runtime-checks
Enable internal UPC runtime checks that validate arguments, and check for inconsistent runtime state. [default=no]

--enable-upc-runtime-stats
Enable internal UPC runtime statistics collection support; these statistics count the number of various significant internal operations, and dump those counts into a per-process statistics file. [default=no]

--enable-upc-runtime-trace
Enable internal UPC runtime trace collection support; a runtime trace is a time stamped log that records various significant internal events; this trace is written to a per-process log file. [default=no]

--enable-upc-runtime-debug
Enable UPC runtime debugging mode, where more expensive internal checks are implemented, and conservative algorithms are used that reduce the degree of parallelism, and that exercise less complex/sophisticated operations provided by the operating system and/or the network communication packages called by the UPC runtime. In addition, conservative compilation options will be used to build the runtime, and debugging symbols will be generated. [default=no]

--enable-upc-link-script
Enable UPC's use of a custom linker script; this will define the UPC shared section as a no load section on targets where this feature is supported (requires GNU LD). [default=yes]

--with-upc-runtime=*MODEL*
Specify the runtime implementation model for UPC, where MODEL may be: *SMP* (Symmetric Multiprocessing) or *Portals4*. [default=*SMP*]

--with-upc-pts=*{struct,packed}*
Choose the representation of a UPC pointer-to-shared. [default=packed]

--with-upc-pts-vaddr-order=*{last,first}*
Choose position of the address field used in the UPC pointer-to-shared representation. [default: first]

--with-upc-pts-packed-bits=*phase,thread,vaddr*
Choose bit distribution in the packed UPC pointer-to-shared representation. [default: 20,10,34]

--enable-upc-triggered-runtime-ops
Enable UPC runtime support for Portals4 triggered operations. [default=yes]

--enable-upc-node-local-mem
Enable UPC runtime support optimization for accessing shared memory of the node local threads. [default=yes]

--with-portals4=*PATH*
Specify prefix directory for installed Portals4 library package. Equivalent to *--with-portals4-include=PATH/include* plus *--with-portals4-lib=PATH/lib*.

--with-portals4-include=*PATH*
Specify directory for installed Portals4 include files.

--with-portals4-lib=*PATH*
Specify directory for the installed Portals4 library.

--with-upc-runtime-pte-base=*BASE*
Specify the base index of the first Portals4 PTE used by the UPC runtime. [default=16]

--with-upc-runtime-max-locks=MAX_LOCKS
Specify the maximum number of locks that can be held by a single UPC thread. [default=1024]

--with-upc-runtime-bounce-buffer-size=*SIZE*
Specify the size (in bytes) of the bounce buffer that is used by the UPC runtime to buffer network data. [default=256K]

--with-upc-runtime-tree-fanout=*WIDTH*
Specify the maximum number of children in each sub-tree used to implement UPC collective operations (e. g., upc_barrier and upc_global_alloc). [default=2]

--with-upc-node-local-mem=SHMEM
Specify type of shared memory used for node local memory accesses. Possible options are "posix" for POSIX Shared Memory or "mmap" for file based mmap-ed memory. [default=posix]

--with-upc-job-launcher=*LAUNCHER*
Specify the job launcher for GUPC runtime. Possible options are "slurm" for the SLURM resource manager, or "yod" for the Portals4 launcher. [default=slurm]

--with-upc-memory-page-size=*SIZE*
Size of the virtual memory page on the target system. Used by threads at system startup to access every page of the local shared memory. [default=4096]

Chapter 5

UPC Program Compilation

GNU UPC is an extension to the GNU Compiler Collection distributed by the Free Software Foundation. In addition to the compile options specified here, all of the normal options listed in the man pages for gcc are available.

The GNU UPC compiler is integrated with the GCC compiler. The compiler processes input files through one or more of four stages: pre-processing, compilation, assembly, and linking.

Suffixes of source file names indicate the language and kind of processing to be done:

file.upc
: UPC source; pre-process, compile, assemble

file.upci
: Pre-processed UPC source; compile, assemble

file.h
: Pre-processor header file; not usually named on command line

file.c
: Files will be compiled as UPC source, unless preceded by -x c

file.i
: Pre-processed source code; compile, assemble

file.s
: Assembler source files; assemble

Files with other suffixes are passed to the linker. Common cases include:

file.o
: Object file

file.a
: Archive file

Linking is always the last stage unless you use one of the -c, -S, or -E options to avoid linking. Compilation errors also stop the process, and the linker is not invoked. For the link stage, all .o files correspond to source files, and all -l options correspond to libraries. Named .o object files, .a archives, and any file names unrecognized by gupc are passed to the linker in command-line order.

5.1 Number of Threads

Within a UPC program, the special symbol *THREADS* refers to the number of parallel execution threads. On each thread, the special symbol *MYTHREAD* refers to the thread number. The number of threads in a UPC application can be specified statically at compile-time or dynamically at execution time. Generally, the number of threads should not exceed the number of available physical central processing units or cores.

If the number of threads is specified statically at compile-time, the special symbol *THREADS* is a constant and can be used freely in any context where a constant is required by the C language specification (for example, in array dimensions in an array declaration). See the *-fupc-threads-N* compilation option.

If the number of threads is specified dynamically at execution time, the special symbol *THREADS* is assigned at run-time, and *THREADS* can be used in array declarations only if the array is qualified as shared and only if one and only one of the shared array's dimensions is specified as an integral multiple of *THREADS*. See the *-fupc-threads-N* execution option.

5.2 Invoking GNU UPC

```
gupc [options] files
```

5.3 GNU UPC Options

gupc accepts the following UPC-specific options.

5.3.1 Information Options

-v
Print (on standard error output) the commands executed to run the stages of compilation. Also print the version number of the compiler driver program.

--version
Print the GNU UPC version number.

5.3.2 Language Options

-x upc
All source files ending in .upc, .c, or .upci will be compiled by the gupc compiler. The -x upc option tells the compiler to process all of the following file names as UPC source code, ignoring the default language typically associated with filename extensions.

-fupc-threads-N
Specify the number of threads at compile-time as N. See the Number of Threads section, above.

5.3.3 Debugging Options

-g
Produce symbolic debugging information.

-dwarf-2-upc
Generate UPC-specific symbolic DWARF-2 debugging information. This debugging information is processed by UPC-aware debuggers including GDB-UPC, a variant of the GDB debugger, and the commercially available Totalview debugger.

5.3.4 Optimization Options

-O0, -O1, -O2, -O3
Specify the optimization level. Nearly all GCC supported optimizations are performed.

Chapter 6

Program Execution

Execution of the compiled program with Portals4 support requires the Portals 4 Reference Implementation Library. Both the Portals4 shared library and *yod* job launcher are required to successfully run the GNU UPC program compiled for Portals4.

By default the Portals 4 Reference Implementation Library installs in the */usr/local* directory. For most of the systems */usr/local/bin* and */usr/local/lib* are already added by the system to the user's execution and library paths. However, if the Portals4 library is installed in a different place (e.g. */usr/local/gupc-p4*) access to the shared libraries and *yod* job launcher must be provided. There are two recommended methods for identifying the location of the Portals4 library, prior to running a linked UPC program:

1. Add the location of the Portals4 library to the *LD_LIBRARY_PATH* environment variable. For example,

   ```
   LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/local/gupc-p4/lib"
   export LD_LIBRARY_PATH
   ```

2. As system administrator add an entry into the system's shared library configuration directory. For example (Fedora Core x86_64):

```
su root
echo '/usr/local/gupc-p4/lib' > /etc/ld.so.conf.d/portals4-x86_64.conf
chmod a-w /etc/ld.so.conf.d/portals4-x86_64.conf
ldconfig
exit
```

6.1 Running the program with *srun*

By default, the GUPC runtime is configured to work with the SLURM resource manager. For example:

```
srun -n 8 --ntasks-per-node=2 program
```

More information on SLURM can be found at https://computing.llnl.gov/linux/slurm/.

6.2 Running the program with *yod*

To use *yod* for program launching, GUPC must be configured with *--with-upc-job-launcher=yod* configure command option.

Also, make sure that the *yod* job launcher is on your PATH. For example if your default shell is bash:

```
export PATH="/usr/local/gupc-p4/bin:$PATH"
```

Hydra Program Manager must be set up to allow for *yod -n N executable*, where *N* is number of threads to spawn, command to properly launch the executable over the InfiniBand network.

More information on Hydra PM can be found at http://wiki.mcs.anl.gov/mpich2/index.php/Using_the_Hydra_Process_Manager.

6.2.1 SSH launcher

Please add the following environment variables to use Hydra's SSH launcher:

```
export HYDRA_HOST_FILE=/path/to/nodes/hostsfile
export HYDRA_LAUNCHER=ssh
```

The host file defined by the *HYDRA_HOST_FILE* defines the compute nodes (hosts) used for the program execution. For example:

```
% cat /path/to/nodes/hostsfile
thor1
thor2
thor3
thor4
```

A simple invocation of a UPC program is shown below.

```
yod -n N upc_program
```

where *N* is the number of UPC threads (i. e., the value of *THREADS*) to instantiate.

A file containing the compute nodes list can also be specified on the *yod* command line:

```
yod -f hostsfile -n N upc_program
```

The compute nodes can also be specified on the *yod* command line:

```
yod -hosts thor1,thor2 -n N upc_program
```

6.2.2 SLURM Launcher

As *yod* auto detects the SLUM resource manager, a UPC program can be executed in the SLURM environment. For example, *salloc* can be used to allocate resources for the UPC program:

```
salloc -n 8 yod upc_program
```

By using yod, a UPC program can also be used in the SLURM batch scripts.

Above, *yod* option for number of threads is not needed as it is acquired from the SLURM allocation.

When executing in the SLURM environment, *HYDRA_HOST_FILE* environment variable must not be set. Also, there is no need for *HYDRA_LAUNCHER=slurm* environment variable.

6.2.3 Program Exit Code

The exit code from the UPC application program is provided to the user as a result of invoking the *yod* job launcher.

6.2.4 Program Arguments

Additional application program arguments can be specified on the *yod* command line right after the name of the program. For example:

```
yod -n 16 upc_program arg1 arg2 ...
```

6.2.5 YOD Options

The *yod* job launcher provides the following options:

-n
Specify the number of threads to run. Note that number of specified *yod* threads must match the number of statically compiled UPC threads.

-hosts
Specify the list of compute nodes to execute on.

-f hostfile
Specify the file containing the list of compute nodes.

To get more information on other *yod* options use the following command:

```
yod --help
```

6.3 Environment Variables

The following environment variables will affect UPC program execution.

UPC_SHARED_HEAP_SIZE
UPC_SHARED_HEAP_SIZE sets the maximum amount of shared heap (per UPC thread) for the program. The default is 256MB per UPC thread. The provided heap size value is optionally multiplied by a scaling factor. Valid scaling factor suffixes are: *K* (Kilobytes), *M* (Megabytes), *G* (Gigabytes), and *T* (Terabytes). For example, to allocate the heap size of one (1) Gigabyte:

```
bash
  export UPC_SHARED_HEAP_SIZE=1G


csh
  setenv UPC_SHARED_HEAP_SIZE 1G
```

TMP , TMPDIR
A path to use for file based mmap-ed node local memory access optimization. By default */tmp* is used.

UPC_NODE_LOCAL_MEM
Disable node local memory access optimization by setting this environment variable to *0*. Useful for debugging purposes only.

UPC_FORCETOUCH
Disable startup page by page access of the local shared memory by setting this environment variable to *0*. Page by page memory touch ensures the correct memory affinity among threads running on the same node. Useful for faster startup time on systems with only one thread per node.

Chapter 7

Node Local Memory Access Optimization

The GUPC for Portals4 runtime supports node local memory access optimization. Access to shared memory of threads on the same node is performed via direct memory access instead of Portals4 PUT/GET routines. Portals4 library is used to determine which threads reside on the same node.

The node local memory access supports the following options:

POSIX Shared Memory
: POSIX shared memory is used to map and access other threads shared memories. POSIX shared objects are named as *upc-mem-THREADID-PID*. This is the default configuration.

MMAP
: File based mmap-ed memory is used to map and access other threads shared memories. To activate this option specify *--with-upc-node-local-mem=mmap* as the GUPC configuration option. By default files are created under */tmp* directory. This can be changed in the execution time by specifying the desired path with *TMP* or *TMPDIR* environment variables. Files are named in a similar fashion as POSIX shared objects.

Node local memory access optimization can be disabled in the configuration time by specifying *--disable-upc-node-local-mem* option or by setting the environment variable *UPC_NODE_LOCAL_MEM=OFF* in the execution time.

Chapter 8

Debug Logging

GNU UPC configured for Portals4 runtime provides support for logging of specific runtime/system events (e.g. accesses to the shared memory). Logging is enabled through a set of environment variables that are set to a list of "facilities" that have debugging output logged.

8.1 Logging Environment Variables

The following environment variables control the logging capabilities of the Portals4 GNU UPC runtime:

UPC_DEBUG
If set, specifies a list of "facilities" that will have debugging output logged.

UPC_DEBUGFILE
Path of log file where UPC runtime debug logs are written.

UPC_LOG
Specifies a list of "facilities" that will be logged.

UPC_LOGFILE
Path of log file where UPC runtime logs are written.

UPC_NO_WARN
The UPC_NO_WARN variable causes startup warnings (such as those displayed when debugging or tracing is enabled) to be omitted.

UPC_QUIET
UPC_QUIET causes all non-application-generated output to be omitted (including both warnings and the initial display of UPC thread layout).

UPC_POLITE
Yield the processor frequently while spin-locking.

UPC_STATS
Specifies a list of "facilities" for will be logged.

UPC_STATSFILE
Path of log file where UPC runtime statistics are written.

UPC_TRACE
If set, specifies a list of "facilities" that will be traced.

UPC_TRACEFILE
Path of log file where UPC trace logs are written.

For all environment variables above that set a filename path, each appearance of a single % will be substituted with the process pid. Two % signs together escape a single %. Non-existent intermediate directories will be created. As a special case, if the filename is "stdout" or "stderr", then output will be directed to the specified file descriptor. A filename with no % indicates that the file will be shared across all processes.

8.2 Logging Facilities

The following logging facilities are provided:

ADDR
UPC casts to local and access to PTS's.

ALLOC
UPC dynamic memory allocation

BARRIER
UPC barrier/notify/wait operations

BROADCAST
UPC runtime internal broadcast operations

COLL
UPC collectives

INFO
General information, program info.

LOCKS
UPC lock operations

MEM
UPC shared memory accesses

MISC
Miscellaneous functions

PORTALS
Portals operations

SYSTEM
System calls

For convenience, a facility "ALL" is provided to enable logging on all facilities.

ALL
Enable logging for all facilities.

8.3 Logging Examples

To enable logging of all events (e.g. DEBUG/TRACE/LOG) set the following environment variables (bash example):

```
export UPC_DEBUG=ALL
export UPC_TRACE=ALL
export UPC_LOG=ALL
```

All the logging output comes on the screen (stdout).

The following settings enables debug logging for memory accesses and barriers:

```
export UPC_DEBUG="MEM,BARRIER"
```

To redirect debug logging to a file, provide the file name for log:

```
export UPC_DEBUGFILE="/tmp/log"
```

To redirect debug logging to multiple files where each file is associated with the process that runs the UPC thread:

```
export UPC_DEBUGFILE="/tmp/log.%"
```

Log files from the above example will be in the form of "/tmp/log.2345" where "2345" is the process id.

Chapter 9

Problem Reporting

For problems and issues related to install and usage of GNU UPC with Portals4 runtime please send an email to the GNU UPC Maintainers.

Chapter 10

References

10.1 Bibliography

[1] GNU UPC Home page http://www.gccupc.org/

[2] GNU UPC Project page http://gcc.gnu.org/projects/gupc.html

[3] Underwood et al. Portals 4 Specification. Sandia Technical Report. January, 2011.

[4] Keith Underwood et al. Enabling Flexible Collective Communication Offload with Triggered Operations. January, 20007.

[5] William Carlson et al. UPC Language Specifications (V1.2). May 31, 2005.