GNU UPC (GUPC) 5.2.0.1 User Manual

Contents

1 Authors and Revision Information 1

2 Introduction 2

3 Requirements 3

3.1 Supported Platforms 3

3.2 Prerequisites 3

4 Installing GNU UPC (GUPC) 4

4.1 Install from Binary Releases 4

4.2 Source Code Configuration and Build 5

4.2.1 Configuration Options 5

4.2.1.1 General Configuration Options 6

4.2.1.2 UPC Compiler Configuration Options 6

4.2.1.3 UPC Runtime Configuration Options 6

4.2.1.4 UPC Portals4 Specific Runtime Configuration Options 7

4.2.2 Build and Install 8

4.2.2.1 Ubuntu 12.4 8

4.2.3 Configure and Build for the Portals4 Runtime 8

5 UPC Program Compilation 9

5.1 Number of Threads 10

5.2 Invoking GNU UPC 10

5.3 GNU UPC (GUPC) Options 10

5.3.1 Information Options 10

5.3.2 Language Options 10

5.3.3 Debugging Options 11

5.3.4 Instrumentation Options 11

5.3.5 Optimization Options 11

6 Program Execution **12**
6.1 SMP Runtime Program Execution . . . 12
6.1.1 Execution (Runtime) Options . . . 12
6.1.2 Environment Variables . . . 13
6.1.3 Program Termination . . . 13
6.2 Portals4 Runtime Program Execution . . . 14
6.2.1 Running the program with *srun* . . . 14
6.2.2 Running the program with *yod* . . . 14
6.2.2.1 SSH launcher . . . 14
6.2.2.2 SLURM Launcher . . . 15
6.2.2.3 Program Exit Code . . . 15
6.2.2.4 Program Arguments . . . 15
6.2.2.5 YOD Options . . . 15
6.2.3 Environment Variables . . . 16
6.2.4 Node Local Memory Access Optimization . . . 16
7 Debugging Support **18**
7.1 Program Backtrace . . . 18
7.1.1 Backtrace Logs . . . 18
7.1.2 Backtrace Events . . . 19
7.1.3 Backtrace in the SMP runtime environment . . . 20
7.1.4 Backtrace in the Portals4 runtime environment . . . 20
7.1.5 Backtrace Configuration . . . 20
7.1.6 Backtrace Environment Variables . . . 20
7.1.7 Backtrace support for high-end tools (e.g. STAT) . . . 21
7.2 Instrumentation . . . 21
7.3 MPIR debugging support . . . 21
7.4 Portals4 Debug Logging . . . 22
7.4.1 Logging Environment Variables . . . 22
7.4.2 Logging Facilities . . . 23
7.4.3 Logging Examples . . . 24
8 Berkeley UPCR Runtime Integration **25**
9 Change Log **27**
9.1 GUPC 5.2.0.1 . . . 27
9.2 GUPC 4.9.0.1 . . . 27
9.3 GUPC 4.8.0.3 . . . 27
9.4 GUPC 4.8.0.2 . . . 28
9.5 GUPC 4.8.0.1 . . . 28

9.6 GUPC 4.7.0.2 . . . 28
9.7 GUPC 4.7.0.1 . . . 29
9.8 GCC UPC 4.5.1.2 . . . 29
9.9 GCC UPC 4.5.1.1 . . . 29
9.10 GCC UPC 4.3.2.5 . . . 29
9.11 GCC UPC 4.3.2.4 . . . 30

10 Platform Specific Configurations 31

10.1 IBM POWER7 (PERCS) . . . 31
10.1.1 System Considerations . . . 31
10.1.1.1 Compile and run on compute nodes . . . 31
10.1.1.2 Shared memory backed file location . . . 31
10.1.2 Compiler build and install . . . 31
10.1.2.1 Prerequisites . . . 31
10.1.2.2 Configure . . . 32
10.1.2.3 Build and Install . . . 32
10.1.2.4 Compilation and Execution . . . 32
10.1.3 Issues . . . 33

11 Problem Reporting 34

12 References 35

12.1 Bibliography . . . 35

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: 5.2.0.1 (2015/08/16)

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 GUPC compiler extends the capabilities of the GNU GCC compiler.

The GUPC compiler and its associated runtime provide the following features:

- UPC Language Specification version 1.3 compliant
- Based on GNU GCC
- GPL licensed
- Configurable pointer-to-shared representation
- Fast bit packed pointer-to-shared support
- GASP support, a performance tool interface for Global Address Space Languages
- Runtime support for uniprocessor and symmetric multiprocessor systems
- Runtime support for Infiniband based clusters with Portals 4.0 library support
- Support for many large scale machines and clusters in conjunction with Berkeley UPC runtime
- Runtime support for UPC collectives
- Runtime support for UPC thread affinity via Linux scheduling affinity and NUMA package
- Runtime support for the UPC Atomic Memory Operations library defined in the UPC Specification version 1.3.
- Runtime support for the UPC pointer-to-shared castability library defined in the UPC Specification version 1.3.
- Runtime support for the UPC asynchronous shared memory bulk copy operations library defined in the UPC Specification version 1.3.
- Runtime support for UPC thread backtrace
- Runtime support for parallel debugging tools with MPIR capabilities
- Runtime support for the STAT backtrace visualization tool
- Binary packages for x86_64, i686
- Binary packages for Linux Fedora, RHEL, SUSE, Ubuntu, CentOS, Mac OS X

Chapter 3

Requirements

3.1 Supported Platforms

The GUPC toolset is available on the following platforms:

Intel x86_64
Linux 64 bit uniprocessor or multiprocessor systems (RHEL, SUSE, Fedora, CentOS, Ubuntu)

Intel x86_64
Apple Mac OS X system

Intel x86
Linux 32 bit systems (Redhat based distributions)

IBM PowerPC
IBM Power6/Power7/Power8 Linux based systems (including PERCS)

3.2 Prerequisites

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 downloaded from the package repository provided 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 the 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

For systems configured for Infiniband, the Portals 4 Reference Library Implementation must be installed on the system for GNU UPC to build and run.

See the Portals 4 Reference Implementation at http://code.google.com/p/portals4/.

Chapter 4

Installing GNU UPC (GUPC)

As with 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

There are two ways to install the GUPC compiler: (1) Install the binary tar file, and (2) Configure, build, and install from the source release.

4.1 Install from Binary Releases

The GUPC binary release is provided in the form of a gzip'ed tar file for the following systems:

- SUSE 11.4, x86_64
- Ubuntu 12.4, x86_64
- RHEL 6.5, x86_64
- CentOS 6.6, i686
- Fedora Core 20, x86_64
- Apple, MacOS X 10.10, x86_64

The gzip'ed tar files contain an installable binary release of the UPC compiler, built for their respective target platforms. For more information on the binary releases please visit GUPC web site: http://www.gccupc.org/gnu-upc-info/gnu-upc-install-from-binary-release

All the binary releases are built with the following configuration options:

- Packed pointer-to-shared representation
- UPC thread affinity supported

The binary releases are built to install under /usr/local/gupc.

The tar file contains paths which do not begin with "/". They are relative to the root directory. To install in /usr/local/gupc, issue the following commands (the Linux Intel x86_64 release is illustrated below):

```
% cd /
% tar xpf upc-binary-release-file.tar.gz
```

The commands above, must be issued from a sysadmin account that has write access to /usr/local. A /usr/local/gupc directory will be created.

If you do not have sufficient privileges to write to the /usr/local directory, you may install and run the compiler somewhere else. Here's an example, where the binary installation file is downloaded into the /upc/test directory. The compiler is the installed in /upc/test/usr/local/gupc:

```
% cd /upc/test
% rm -rf usr/local/gupc
% tar xf upc-5.2.0.1-x86_64-linux-fc20.tar.gz
% cat > count.upc << EOF
#include <upc.h>
#include <stdio.h>

int
main ()
{
  int i;
  for (i = 0; i < THREADS; i++)
    {
      if (MYTHREAD == i)
        {
          printf ("%d ", i + 1);
        }
    }
}
EOF
% /upc/test/usr/local/gupc/bin/upc count.upc
% a.out -fupc-threads=5
1 2 3 4 5
```

4.2 Source Code Configuration and Build

Configuring and building GNU UPC is similar to configuring GCC itself. The following discussion provides some guidance and help in building and installing GNU UPC, as well as describing options that are GNU UPC specific.

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

It is a requirement that GUPC be built into a separate directory from the sources which does not reside within the source tree. This is how generally GNU GCC is also built.

When configuring GUPC, 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 GUPC in the same directory, run 'make distclean' to delete all files that might be invalid. One of the files that this step 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.

The simplest command to configure GUPC looks like this:

```
% mkdir objdir
% cd objdir
% $srcdir/configure [options] --prefix=/usr/local \
        --enable-languages=c,c++
```

By default, the SMP based runtime is configured and built.

4.2.1 Configuration Options

The following GCC and GUPC options are provided to better tailor GUPC 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

4.2.1.1 General Configuration Options

--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 GUPC we recommend */usr/local/gupc*. [default: /usr/local]

--[enable|disable]-bootstrap
By default, GUPC will be built in three stages, where in the last stage the built compiler compiles itself. Bootstrapping is a useful method of verifying that the compiler is operational, but it takes three times as long to build the compiler. Specifying --disable-bootstrap reduces build time to 1/3 of the default build time. [default: enabled]

--[enable|disable]-checking
Primarily intended as an aid to developers, the checking switch enables various internal checks within the GUPC compiler. Compilations will be slower, but the checking can help catch bugs in the compiler's internal logic. [default: disabled]

--[enable|disable]-multilib
Build alternate library versions (e.g. 32-bit libraries on the 64-bit system). [default: enabled]

4.2.1.2 UPC Compiler Configuration Options

--[enable|disable]-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: enabled]

--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-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]

4.2.1.3 UPC Runtime Configuration Options

--with-upc-runtime=*MODEL*
Specify the runtime implementation model for UPC, where MODEL may be: *SMP* (Symmetric Multiprocessing) or *Portals4* (Infiniband with Portals 4.0 Reference Library). [default=*SMP*]

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

--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). [default: 4]

--[enable|disable]-upc-backtrace
Enable stack frame backtrace report when UPC runtime fatal errors occur or by user request (via signal) [default: enabled]

--[enable|disable]-upc-backtrace-gdb
Enable the use of GDB for UPC stack backtrace [default: enabled]

--[enable|disable]-upc-backtrace-signal
Enable signal support for UPC stack backtrace [default: enabled]

--with-upc-backtrace-gdb=GDB
Specify which GDB to use for UPC backtrace support [default: gdb]

--with-upc-backtrace-signal=SIGNAL
Specify the signal to be used for UPC stack backtrace [default: SIGUSR1]

4.2.1.4 UPC Portals4 Specific Runtime Configuration Options

--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-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-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-max-outstanding-puts=*SIZE*
Specify the maximum number of outstanding remote put requests. [default=256]

--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]

4.2.2 Build and Install

To build GUPC after the configuration step:

```
% make >make.log
% make install >install.log
```

An optional "-j" argument on 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.

4.2.2.1 Ubuntu 12.4

Ubuntu distribution (version 11.10 and up) integrates support for installing packages from multiple architectures on a single system (https://wiki.ubuntu.com/MultiarchSpec). The current version of GUPC (and GNU GCC) is not compatible with this approach and the following steps must be taken in order to build GUPC on the Ubuntu platform:

- Set the following environment variables

```
export LIBRARY_PATH=/usr/lib/x86_64-linux-gnu/
export C_INCLUDE_PATH=/usr/include/x86_64-linux-gnu
export CPLUS_INCLUDE_PATH=/usr/include/x86_64-linux-gnu
```

- Configure GUPC with the "--disable-multilib" option

More information on this subject can be found in the following e-mail list thread:

http://gcc.gnu.org/ml/gcc-patches/2011-08/msg01649.html

4.2.3 Configure and Build for the Portals4 Runtime

The simplest way to configure and build GUPC for the Portals4 runtime is to specify *portals4* as the runtime model:

```
% $srcdir/configure --enable-languages=c.c++ \
                    --prefix=/usr/local \
                    --with-upc-runtime=Portals4
% make >make.log
% make install >install.log
```

Use the *--with-portals4=* configuration option to specify a path to the Portals4 reference library is located, if it is installed in a non standard place.

By default the GUPC runtime is configured to work with the SLURM resource manager. Use *--with-upc-job-launcher=yod* option if you wish to configure *yod* instead (e.g. for SSH program launching).

Please refer to the Portals4 configuration options section if additional tunning is required.

Chapter 5

UPC Program Compilation

The GUPC compiler 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 the GNU GCC are available.

The GUPC 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 in the compilation process unless you use one of the -c, -S, or -E options to avoid linking. Compilation errors also stop the process; if they occur, the linker is not invoked. For the link stage, all .o files refer to compiled object files, and all -l options refer 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 identifier *THREADS* refers to the number of parallel execution threads. On each thread, the special identifier *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, then *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 runtime, 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] file ...
```

5.3 GNU UPC (GUPC) Options

GUPC accepts the following UPC-specific options.

5.3.1 Information Options

-v
Print the commands executed to run the stages of compilation. Also print the version number of the compiler driver program.

--version
Print the GUPC 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.

-fupc-pthreads-model-tls
Compile for the POSIX threads (pthreads) environment. Each UPC thread is implemented as a pthread.

-fupc-inline-lib
Inline UPC runtime library calls. This option is turned on by default when compiled with optimization and the -fno-upc-inline-lib option must be specified to turn it off. In general, inlining of the runtime library is more efficient but the generated code is more complex. Disabling this capability can sometimes be helpful when debugging the application, or when a compilation code generation error is suspected.

-fupc-pre-include
Pre-include UPC runtime header files. This option is turned on by default. Use -fno-upc-pre-include to disable this pre-include facility.

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.

-fupc-debug
Generate calls to the UPC runtime library that include source filename and line number information that is used to print more informative error messages when errors are detected at runtime.

5.3.4 Instrumentation Options

-fupc-instrument
Instrument UPC shared accesses and library calls using GASP tool support. This option implies -fno-upc-inline-lib.

-fupc-instrument-functions
Instrument functions calls using GASP tool support. This option implies -fupc-instrument and -fno-upc-inline-lib.

5.3.5 Optimization Options

-O0, -O1, -O2, -O3
Specify the optimization level.

Chapter 6

Program Execution

Execution of the UPC program depends on the runtime it is linked with.

6.1 SMP Runtime Program Execution

To execute a UPC program that has been compiled and linked with GUPC SMP runtime simply invoke it with appropriate options. There are several options that are recognized and used by the UPC runtime; these options are specified on the command line when you invoke a UPC program. Before calling the "main()" function of a UPC program, the UPC runtime removes all options that begin with the prefix -fupc- and that immediately follow the UPC program name on the command line.

```
UPC_program [number of threads] [heap size] [affinity options]
            [program arguments]
```

6.1.1 Execution (Runtime) Options

The following runtime options are available:

-fupc-threads-N | -n N
: Specifies, at runtime, the number of parallel execution threads as N. If the UPC program was not compiled with the -fupc-threads=N option, either the -fupc-threads=N or -n N command-line option is required when you invoke the UPC program. See the Number of Threads section.

-fupc-heap-HEAPSIZE
: Specifies the size of the heap available to each thread as HEAPSIZE. A suffix of K indicates that HEAPSIZE is expressed in kilobytes (2^{10} bytes). A suffix of M indicates that HEAPSIZE is expressed in megabytes (2^{20} bytes). A suffix of G indicates that HEAPSIZE is expressed in gigabytes (2^{30} bytes). If a suffix is not present, HEAPSIZE is expressed in bytes. If the -fupc-heap-HEAPSIZE option is not supplied, the runtime system will use a default heap size of 16 megabytes per thread.

The following options specify thread scheduling and Non-Uniform Memory Access (NUMA) policies:

-sched-policy [cpu|strict|node|auto]
: Specifies the scheduling policy for threads. Default is *auto*.

cpu
: specifies that threads are evenly scheduled over available CPUs. (A CPU is a processor with a single core or a core unit in a multicore processor.)

strict
: is similar to cpu scheduling except that one to one mapping of threads and CPUs is required.

node
specifies that threads are scheduled on nodes if a NUMA-aware kernel is available.

auto
specifies that the UPC runtime should not manage scheduling of UPC threads.

-sched-cpu-avoid n1,n2,..
Specifies the availability of CPUs for UPC thread scheduling. The UPC runtime will not schedule any thread on the specified CPUs.

-mem-policy [node,strict,auto]
Specifies the memory allocation policy if a NUMA-aware kernel is available. Default is *auto*.

node
allocates memory first from the node on which a thread is scheduled to run.

strict
allocates memory only from the node on which a thread is scheduled to run.

auto
lets the kernel decide the memory allocation policy.

6.1.2 Environment Variables

The following environment variables will affect UPC program execution.

TMP | TMPDIR
Temporary directory for file based memory mapped shared space. Ideally on a Linux based system this should point to *tempfs* file system. [default: /tmp]

UPC_BACKTRACE
Enable backtrace generation if a fatal error occurs in the UPC program. Set this environment variable to 1 to enable backtrace. [default: disabled]

UPC_BACKTRACEFILE
Template for the backtrace files if explicitly requested by the user. [default: stderr]

UPC_BACKTRACE_GDB
The file path of the GDB debugger that will be used to generate a backtrace. [default: gdb]

6.1.3 Program Termination

The GUPC compiled program completes execution in several ways:

Normal completion
All UPC threads execute a call to the *exit* procedure or return from the main procedure. The exit code from the last UPC thread to exit is reported as the UPC program's exit code. Conflicting exit codes from various UPC threads are reported.

UPC global exit
Upon detecting a UPC thread that exited via *upc_global_exit*, the monitor thread terminates all other UPC threads. The exit code passed as an argument to *upc_global_exit* is returned as the program's exit code.

Abort
Upon detecting a UPC thread that exited via *abort*, the monitor thread terminates all other UPC threads and aborts the UPC program.

Unhandled Signals
Unhandled signal (e.g. SIGTERM, SIGINT) immediately terminates the UPC program. Additionally, sending the SIGTERM signal individually to the monitor thread or any of the UPC threads also terminates the UPC program.

6.2 Portals4 Runtime 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
```

6.2.1 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/.

6.2.2 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"
```

The Hydra Program Manager must be set up to support program invocation (for example, *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.

6.2.2.1 SSH launcher

To use Hydra's ssh based job launcher, add the following environment variables:

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

The host file given by the *HYDRA_HOST_FILE* environment variable describes 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
```

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

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

6.2.2.3 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.

6.2.2.4 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 ...
```

6.2.2.5 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
```

6.2.3 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.

UPC_BACKTRACE
Enable backtrace generation if a fatal error occurs in the UPC program. Set this environment variable to 1 to enable backtrace. [default: disabled]

UPC_BACKTRACEFILE
Template for the backtrace files if explicitly requested by the user. [default: stderr]

UPC_BACKTRACE_GDB
The file path of the GDB debugger that will be used to generate a backtrace. [default: gdb]

6.2.4 Node Local Memory Access Optimization

The GUPC Portals4 based runtime supports node local memory access optimizations. Access to shared memory of threads on the same node is performed via direct memory access instead of Portals4 PUT/GET routines.

The Portals4 based runtime supports two implementation choices for the storage of node local shared memory:

POSIX
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 optimizations 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=0*.

Chapter 7

Debugging Support

7.1 Program Backtrace

The GUPC runtime supports UPC program backtrace (also called stack backtrace or stack traceback). It is used during interactive and/or post-mortem debugging and can be used to determine the sequence of nested functions called up to the point where the backtrace is generated. Program backtrace is available for the SMP based runtime only.

The GUPC backtrace is generated in the following situations:

1. On catastrophic events when the GUPC runtime aborts the running thread. By default, the GUPC backtrace uses GDB to provide detailed information on the thread's stack frames. Only the first thread reaching the abort statement generates a backtrace log.
2. On a specific request from the user. Sending a pre-configured signal to the operating system process associated with a specific UPC thread will cause a backtrace to be generated for that UPC thread. By default, SIGUSR1 is used to signal a backtrace.

By default, backtrace on catastrophic events is disabled, but it can be enabled by setting the "UPC_BACKTRACE" environment variable.

Backtrace via a user request cannot be disabled. However, it can be redirected to a file or to *stderr*. By default *stderr* is used.

7.1.1 Backtrace Logs

There are three kinds of backtrace logs depending on the configuration and the capabilities of the underlying operating system.

DETAILED
GDB's '*bt*' command is used to produce the backtrace with nested procedures and their arguments shown. To get full benefit of this back trace, an executable must be compiled with debugging turned on (e.g. *-g -O0*). Detailed backtrace logs are produced only on catastrophic events.

SIMPLE
The GLIBC *backtrace* capability (http://www.gnu.org/software/libc/manual/html_node/Backtraces.html) and addr2line program are used to produce the source file and line number for each stack frame. addr2line (part of binutils) must be installed when GUPC is configured.

RAW
The GLIBC '*backtrace*' capability is used to produce procedure names and addresses for each stack frame. Depending on the compiler option '*-rdynamic*' (instructs the ELF linker to create a special section with additional symbols) backtrace will generate differing output.

With the *'-rdynamic'* option
Procedure name, addresses, and offsets from the beginning of the procedure is displayed for each frame.

Without the *'-rdynamic'* option
Raw addresses are displayed. The UPC runtime library checks if the *'-rdynamic'* option is supported and adds the appropriate options on the linker command line, if needed.

7.1.2 Backtrace Events

The following events create backtrace logs (depending on the GUPC configuration and environment variable settings):

1. Fatal signal (SIGSEGV, SIGBUS, SIGFPE) or GUPC *runtime* failure
2. User defined signal received by the UPC thread. The user can request a backtrace log by sending a predefined signal to the process that has a UPC thread mapped to it.

By sending a signal to a UPC thread (a process associated with the UPC thread) causes the process to dump backtrace information. The following is an example backtrace output if the addr2line program is present on the system:

```
[   3][0] __upc_wait /path/upc_barrier.upc:295
[   3]       BARRIER ID: 0
[   3][1] __upc_barrier /path/upc_barrier.upc:340
[   3][2] proc0 /path/bt-example.upc:38
[   3][3] upc_main /path/bt-example.upc:61
```

Note that upon detecting a "__upc_wait" procedure in the stack trace, the backtrace code prints the barrier ID on the next output line.

As mentioned above, *gdb* is used to generate a backtrace when catastrophic events are detected. By default a segmentation violation in a UPC program produces the following output:

```
% ./segv-backtrace
./segv-backtrace: UPC error: Segmentation fault.
Aborted (core dumped)
```

However, with backtrace enabled the following output is generated:

```
% UPC_BACKTRACE=1 ./segv-backtrace
./segv-backtrace: UPC error: Segmentation fault.
Thread 0 GDB backtrace:
0x0000003af1eac2ce in __libc_waitpid (pid=<optimized out>, stat_loc=0x0,
options=0) at ../sysdeps/unix/sysv/linux/waitpid.c:32
32             return INLINE_SYSCALL (wait4, 4, pid, stat_loc, options, NULL);
#0  0x0000003af1eac2ce in __libc_waitpid (pid=<optimized out>, stat_loc=0x0,
options=0) at ../sysdeps/unix/sysv/linux/waitpid.c:32
#1  0x000000000040e430 in __upc_fatal_backtrace ()
#2  0x000000000040846d in __upc_fatal ()
#3  0x000000000040e5fc in __upc_fault_handler ()
#4  <signal handler called>
#5  0x0000000000406a65 in proc2 (a=5) at segv-backtrace.upc:5
#6  0x0000000000406a7e in proc1 (a=5) at segv-backtrace.upc:9
#7  0x0000000000406a95 in proc0 (a=5) at segv-backtrace.upc:13
#8  0x0000000000406aa5 in upc_main () at segv-backtrace.upc:21
#9  0x0000000000407d8c in __upc_run_this_thread ()
#10 0x0000000000407e1c in __upc_run_threads ()
#11 0x0000000000408604 in main ()
Aborted (core dumped)
```

7.1.3 Backtrace in the SMP runtime environment

The SMP-based UPC runtime has a *monitor* thread which creates the processes that are mapped to UPC threads and then monitors those processes. Depending on the configuration, sending a backtrace signal to the monitor thread causes the following depending on the value of the UPC_BACKTRACEFILE environment variable.

UPC_BACKTRACEFILE is NOT set
The UPC monitor thread shows the mapping between UPC thread numbers and their system process IDs (pid).

UPC_BACKTRACEFILE is set
The UPC monitor thread sends the signal to all UPC threads to dump their backtrace files. The location of the trace files depends on the value of the UPC_BACKTRACEFILE environment variable. By default, files are dumped in the current directory with file names in the form of "backtrace.THREAD-ID".

7.1.4 Backtrace in the Portals4 runtime environment

The Portals4-based UPC runtime also supports the backtrace.

Use SLURM's *scancel* command to send a request for a backtrace to all threads in the specified job. For example:

```
% squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
   3594        ib   test19    nenad   R       0:12      1 thor1
% scancel --signal=USR1 3594
```

7.1.5 Backtrace Configuration

The following configuration options are provided to control the backtrace behavior in the GUPC runtime:

--enable-upc-backtrace
Enable/disable backtrace. [default=enabled]

--enable-upc-backtrace-gdb
Enable/disable usage of GDB for backtrace on catastrophic events. [default=enabled]

--with-upc-backtrace-gdb=[path-to-gdb]
Specify the GDB program to use to generate a catastrophic backtrace report. [default: gdb]

--enable-upc-backtrace-signal
Enable backtrace via user initiated signal. [default=enabled]

--with-upc-backtrace-signal=[SIGNAL]
Use the specified *'SIGNAL'* for the backtrace requests. [default=SIGUSR1]

Note
Use of GLIBC backtrace capability is disabled if GLIBC does not support backtrace.

7.1.6 Backtrace Environment Variables

UPC_BACKTRACE
Enable backtrace for runtime fatal events. By default backtrace logging on fatal events is disabled (event though it may be configured).

UPC_BACKTRACE_GDB=[path-to-gdb]
Override the configured GDB for backtrace logging (e.g. UPC_BACKTRACE_GDB=/usr/local/bin/gdb).

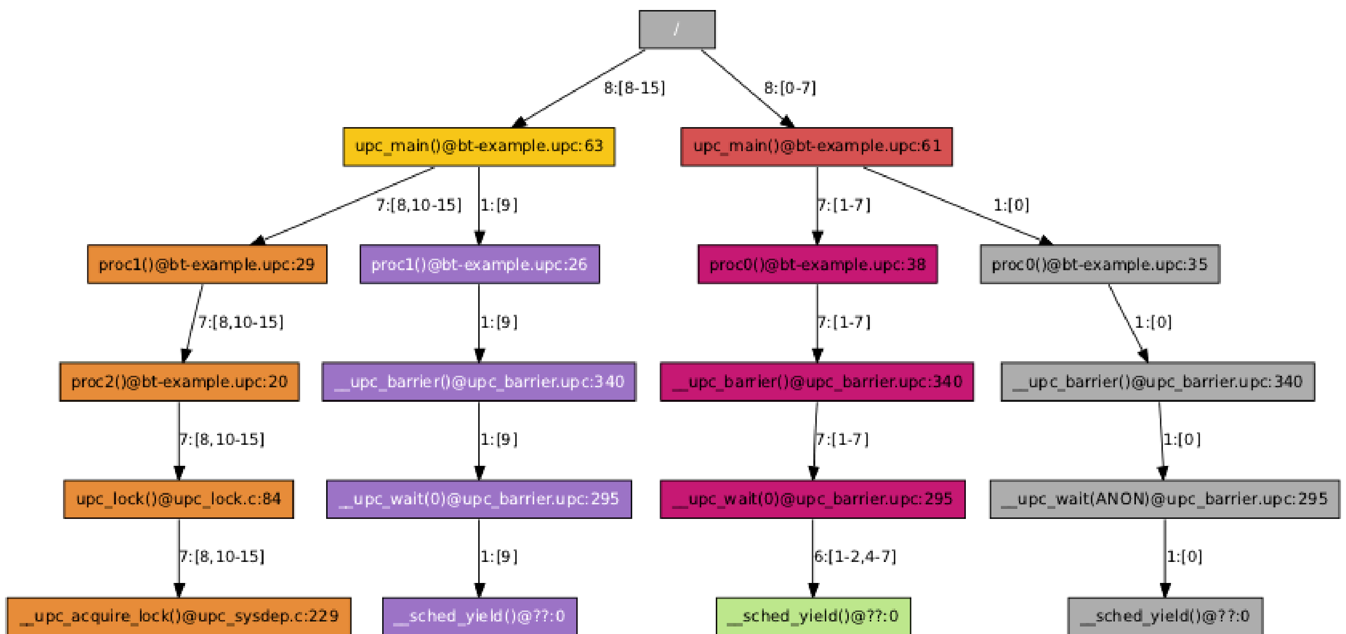
UPC_BACKTRACEFILE="file-prefix"
If specified, backtrace lines are written to the files with the specified prefix. These files have names with the following form: *file-prefix*.THREAD-ID. By default (if no *file-prefix* is specified) the string "backtrace" is used.

7.1.7 Backtrace support for high-end tools (e.g. STAT)

By defining the *UPC_BACKTRACEFILE* environment variable, the user can redirect backtrace logs to a file instead of the screen (*stderr*). If an empty *UPC_BACKTRACEFILE* value is given, the backtrace log file names will in the form of "backtrace.MYTHREAD", Otherwise, *UPC_BACKTRACEFILE* is used as a file prefix to direct all files to the specified directory (e.g. if set to "/tmp/trace", backtrace log files will be in the form of "/tmp/trace-PID.MYTHREAD").

Backtrace log files can be used by the stat-merge python script developed by the GUPC and STAT developers.

The following image represents the result of the backtrace results created by the STAT tool.



7.2 Instrumentation

The GUPC compiler and runtime support instrumentation of UPC shared accesses and UPC runtime library calls, as well as instrumentation of all user-specified function calls.

Instrumentation is enabled by providing the "-fupc-instrument" and "-fupc-instrument-functions" options on the UPC program command line. Both of these options imply -fno-upc-inline-lib.

7.3 MPIR debugging support

The GUPC compiler runtime supports the MPIR parallel debugging protocol as specified by the MPI Forum Working Group on Tools: http://www.mpi-forum.org/docs/mpir-specification-10-11-2010.pdf

The following MPIR debugging options are supported:

1. Start the UPC program with a parallel debugger tool
2. Attach to an already running UPC program

The following MPIR variables/functions are present in the GUPC runtime:

VOLATILE int MPIR_being_debugged
MPIR_being_debugged is an integer variable that is set or cleared by the tool to notify the starter process that a tool is present.

MPIR_PROCDESC *MPIR_proctable
MPIR_proctable is a pointer variable set by the starter process that points to an array of MPIR_PROCDESC structures containing MPIR_proctable_size elements. This array of structures is the process descriptor table.

int MPIR_proctable_size
MPIR_proctable_size is an integer variable set by the starter process that specifies the number of elements in the procedure descriptor table pointed to by the MPIR_proctable variable.

VOLATILE int MPIR_debug_state
MPIR_debug_state is an integer value set in the starter process that specifies the state of the MPI job at the point where the starter process calls the MPIR_Breakpoint function.

VOLATILE int MPIR_debug_gate
MPIR_debug_gate is an integer variable that is set to 1 by the tool to notify the MPI processes that the debugger has attached.

void MPIR_Breakpoint() {}
MPIR_Breakpoint is the subroutine called by the starter process to notify the tool that an MPIR event has occurred.

int MPIR_partial_attach_ok
MPIR_partial_attach_ok is a symbol of any type (preferably int) that informs the tool that the MPI implementation supports attaching to a subset of the MPI processes.

7.4 Portals4 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.

7.4.1 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.

7.4.2 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.

7.4.3 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 8

Berkeley UPCR Runtime Integration

The GUPC compiler can be used to compile UPC programs which are linked the GASNet based UPC runtime (called UPCR) developed by Berkeley (LBNL). The GUPC and Berkeley (UPCR) runtime combination is available on all platforms supported by GUPC. Use of the UPCR runtime increases the range of communication methods that can be used to implement UPC remote access and synchronization primitives. More information on the Berkeley UPCR project site at: the UPCR web site http://upc.lbl.gov/.

Follow these steps to build GUPC with the Berkeley UPCR runtime support.

1. Download the latest GUPC release and follow the installation instructions
2. Download the latest Berkeley UPCR runtime and follow the instructions on building UPCR with GUPC support. Please consult the INSTALL.txt document on specifics of the port.

This small example demonstrates the process of integrating the GUPC compiler and the Berkeley runtime. For simplicity, a hypothetical directory structure under */upc* will be used for building both the GUPC compiler and the Berkley UPCR runtime.

> **Note**
> Source release tar files for both the GUPC and the Berkeley runtime unpack in their respective top level directories (e.g. gnu-upc-5.2.0.1). Please replace "unpacked-gupc-dir" and "unpacked-upcr-dir" with the correct directory names in the example bellow.

- Configure and build the GUPC compiler.

```
% mkdir /upc/gupc
% cd /upc/gupc
% mkdir src bld rls
% cd src
% tar xfz gupc-source-tar-file.tar.gz
% cd ../bld
% ../src/unpacked-gupc-dir/configure \
  --enable-languages=c,c++ --prefix=/upc/gupc/rls
% make -j 8 >make.log
% make install >install.log
```

- Verify that the GUPC compiler is operational.

```
% /upc/gupc/rls/bin/upc --version
[...]
upc (GCC) 5.2.0 20150816 (GNU UPC 5.2.0-1)
[...]
```

- Configure and build the Berkeley UPCR toolset and runtime.

```
% mkdir /upc/upcr
% cd /usr/upcr
% mkdir src bld rls
% cd src
% tar xfz upcr-source-tar-file.tar.gz
% cd ../bld
% ../src/unpacked-upcr-dir/configure GUPC_TRANS=/upc/gupc/rls/bin/upc \
        --prefix=/upc/upcr/rls \
        --with-multiconf=+dbg_gupc,+opt_gupc
% make -j 8 >make.log
% make install >install.log
```

- Verify that the Berkeley UPCR toolset is operational by checking that line "Translator location" of the compiler output contains the GUPC compiler specified during the configuration step.

```
% /upc/upcr/rls/bin/upcc -gupc -V
[...]
----------------------+------------------------------------------------------
 UPC-to-C translator  | 5.2.0.1, built on Oct 23 2013 at 01:01:00
----------------------+------------------------------------------------------
 Translator location  | /usr/local/gupc/bin/upc
----------------------+------------------------------------------------------
[...]
```

Chapter 9

Change Log

9.1 GUPC 5.2.0.1

- GCC 5.2 based
- Bug fixes
  - Allow for -flto switch
  - #pragma upc c_code does not undefine UPC keywords
  - Fix 128-bit struct poiter to shared atomic get/set/cswap
  - Fix internal compiler error on nested procedures
  - GUPC accepts the -fupc-threads=N switch
  - Fix static initialization of __int128 causes internal compile error
  - Comparison with (shared void *) does not ignore phase using packed PTS representation
  - Comparison between pointers-to-shared that differ only in target blocking factor not diagnosed as an error

9.2 GUPC 4.9.0.1

UPC Specification 1.3 Related Changes

- Barrier statements now accept any expression that can be converted to an integer
- The Atomic Memory Operations (AMO) library is supported
- The UPC castable pointers-to-shared library is supported
- The non-blocking shared memory bulk operations library is supported
- Conversions between UPC pointers-to-shared and integers are supported.
- upc_types.h is defined as a separate #include file

9.3 GUPC 4.8.0.3

- Correct the bootstrap build on the PowerPC platform
- Correct the build of dependencies in the UPC runtime libraries
- Correct the UPC data ordering in shared string handling functions in the Portals 4.0 runtime
- Portals 4.0 runtime requires *-lpthread* on the link command line for the systems that use the newer versions of the gld (gold)

9.4 GUPC 4.8.0.2

- Upgrade to the latest Portals 4.0 runtime

9.5 GUPC 4.8.0.1

- Compiler upgrade to GCC 4.8 branch
- Implement various UPC version 1.3 changes
  - upc_tick wall-clock timer library
  - upc_all_free and upc_all_lock_free collective shared memory de-allocation
  - deprecate upc_local_alloc
- Add GCC compatible GUPC command line driver
- Improve GUPC man pages and documentation
- Add Infiniband and Portals 4.0 support
- Add program backtrace support
- Add integration with STAT tool
- Add MPIR debugging interface option to allow for collective debug session start
- UPC barrier optimization using a tree based barrier algorithm
- UPC lock optimization using MCS lock algorithm

9.6 GUPC 4.7.0.2

- Support the use of relative paths to the ‘configure’ command
- Disallow configuring GUPC with the --enable-shared switch. UPC programs must be linked with the static version of libgupc.
- Fix a build error that occurred when the UPC language dialect is omitted from the --enable-languages switch
- Fix the build to enable the make of "profiledbootstrap"
- Fully support GUPC builds with the --program-suffix switch
- upc_addrfield() now returns a consistent value across all GUPC configurations (packed/struct, UPC link script)
- Correct the GCCUPC Config information encoded in a UPC executable program
- Fix the runtime to allow a mixture of programs compiled with static and dynamic number of threads. The static value takes precedence.
- Fix a crash in the gupc driver when certain invalid command switches were specified
- Fix compiler generated calls to the profiling access routines when -fupc-debug is specified
- Ensure that the correct file/line number is passed to the profiling access routines when -fupc-debug is specified
- Disable calls to the profiling access routines inside the UPC compiler generated shared variable initialization procedures.

9.7 GUPC 4.7.0.1

- Upgrade to GCC 4.7 baseline

9.8 GCC UPC 4.5.1.2

- Maintenance release of GCC UPC
- Improved error diagnostics
- Fix segfault on incomplete array definition
- Provide thread safe rand() function for GCC UPC runtime

9.9 GCC UPC 4.5.1.1

- Upgrade to GCC 4.5.1 baseline
- Improved runtime error message reporting. This capability· is enabled by the newly introduced -fupc-debug switch.
- The following errors were corrected in this version of the compiler:
- Layout qualifier within a typedef is not incorporated into the referencing type
- Attempt to use a block size that exceeds maximum is not explicitly diagnosed upc_forall with empty clauses mis-diagnosed as syntax error
- Nested upc_forall() semantics are not implemented
- Static initializers which reference the address of a shared variable are unsupported
- Failure when attempting to specify maximum blocksize in dynamic threads environment
- Shared array exceeds maximum size on 32-bit hosts
- Error message is off-by-one when given blocksize is greater than UPC_MAX_BLOCKSIZE (ILP64 struct sptr)
- Failure on *[*]* layout factor on multi-dimensional shared array with dynamic threads
- Failure on *[*]* layout factor applied to array with static threads and size not a multiple of threads
- Failure to initialize per-thread static variables that refer to shared addresses

9.10 GCC UPC 4.3.2.5

- Added support for NUMA API 2.0
- Disabled link script support for Apple Mac OS X
- Documented -fupc-instrument[-functions] switch
- Added debugging support for packed shared pointer representation
- Upgraded binary releases to the latest OS versions

9.11 GCC UPC 4.3.2.4

- Support for the Apple Mac OS X platform
- Improved conformance to the UPC language specification (version 1.2)
- UPC collectives support implemented in the GCC UPC SMP-based runtime
- GASP (a performance analysis tool interface for Global Address Space
- Programming models) support implemented in both the GCC UPC compiler and the GCC UPC SMP-based runtime
- Code optimization improvements as a result of moving to the GCC 4.3 baseline

Chapter 10

Platform Specific Configurations

10.1 IBM POWER7 (PERCS)

POWER7 is Power Architecture based symmetric multiprocessor designed and built by IBM. PERCS (Productive, Easy-to-use, Reliable Computing System) is a system based on POWER7 architecture. Each compute node reports 125 processor cores which makes it suitable for the GUPC SMP based runtime.

10.1.1 System Considerations

10.1.1.1 Compile and run on compute nodes

The PERCS system makes the distinction between login and compute nodes. It is possible to build the GUPC compiler on the compute node (with a proper job scheduler reservation and no interference to other users). In this case, make sure that the compute node has the required pre-requisite packages.

Use this LoadLeveler command to reserver a compute node for GUPC testing. The command bellow reserves on enode for 120 minutes starting in 5 minutes.

```
llmkres -t ‘date --date="+ 5 minutes" +"%D %k:%M"‘ -d 120 -n  1
```

Find the reserved node with the following command:

```
llqres -l -u $USER
```

10.1.1.2 Shared memory backed file location

For the optimal runtime performance make sure that the file used for backing the UPC shared memory resides on a main memory backed file system (tempfs). For example, set your *TMP* or *TMPDIR* environment variable to */dev/shm* if default file system (*/tmp*) is not mounted on a *tempfs* device. By default the GUPC runtime uses */tmp* for the shared memory backed file.

10.1.2 Compiler build and install

10.1.2.1 Prerequisites

Make sure that all GNU GCC prerequisites are installed on the system. For the RHEL 6.2 the following packages are needed:

- gmp, gmp-devel
- mpfr, mpfr-devel

- libmpc, libmpc-devel

If for some reason the above packages are not installed, you can download them into the GUPC source directory and configure/build them as part of the compiler build. The GUPC provided script *download_prerequisites* accomplishes this by downloading gmp, mpfr, and mpc source packages and unpacking them under the GUPC source tree.

```
% cd src
% ./contrib/download_prerequisites
```

Please visit the GCC prerequisites information page if your are considering adding some additional features: http://www.gccupc.org/-gnu-upc-info/gnu-upc-prerequisites

For best performance, these NUMA related packages are also required:

- numactl, numactl-devel

10.1.2.2 Configure

The GUPC compiler for the POWER7 architecture is configured and built in the same manner as on other systems. However, some additional configuration options are recommended:

--with-cpu-64=power7
: Perform code generation for the POWER7 architecture.

--with-cpu-32=power4 --with-tune-32=power6
: Optimize multilib support when compiling in 32-bit mode.

--with-long-double-128
: Make the *long double* type 128 bits for compatibility with other systems.

The recommended GUPC configure command takes the following form:

```
../src/configure --prefix=PATH-TO-RELEASE-DIR \
        --with-languages=c,c++ \
        --with-cpu-64=power7 \
        --with-cpu-32=power4 \
        --with-tune-32=power6 \
        --with-long-double-128
```

10.1.2.3 Build and Install

Once the GUPC toolset is configured, run make to build and install in the configured install area (specified with --prefix switch).

```
% make -j 32
% make install
```

The GUPC build supports a parallel make process. The number of processes used for a parallel make depends on the node where the make command executes. On a login node, it is generally recommended that the the number of parallel make processes is limited to avoid contention with other users. On a dedicated compute node, all available processors can be used.

10.1.2.4 Compilation and Execution

To compile and execute the GUPC compiled program simple execute it.

```
% gupc -o test test.upc
% ./test -n 64
```

The compile phase can be run on either the login or compute node.

10.1.3 Issues

/tmp is small, or not *tempfs*
Set the *TMP* environment variable to */dev/shm* before running UPC programs.

multilib build for *soft-fp* produces many warnings
While building libgcc some noisy warnings in the form of "warning: no previous prototype for" appear. These can be safely ignored.

Chapter 11

Problem Reporting

For problems and issues related to the installation and use of GUPC please send an email message to GUPC Support.

For problems and issues related to the Portals P4 UPC runtime please use the issue tracker on the portals-upc Google project page.

Chapter 12

References

12.1 Bibliography

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

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

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

[4] UPC Specification Group UPC Language Specifications (V1.3). 2012. http://code.google.com/p/upc-specification/

[5] MPI Forum Working Group on Tools: The MPIR Process Acquisition Interface. Oct 11, 2010. http://www.mpi-forum.org/docs/mpir-specification-10-11-2010.pdf