Darshan-runtime installation and usage

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Introduction

This document describes darshan-runtime, which is the instrumentation portion of the Darshan characterization tool. It should be installed on the system where you intend to collect I/O characterization information.

Darshan instruments applications via either compile time wrappers for static executables or dynamic library preloading for dynamic executables. An application that has been instrumented with Darshan will produce a single log file each time it is executed. This log summarizes the I/O access patterns used by the application.

The darshan-runtime instrumentation only instruments MPI applications (the application must at least call `MPI_Init()` and `MPI_Finalize()`). However, it captures both MPI-IO and POSIX file access. It also captures limited information about HDF5 and PnetCDF access. Darshan also exposes an API that can be used to develop and add new instrumentation modules (for other I/O library interfaces or to gather system-specific data, for instance), as detailed in this document. Newly contributed modules include a module for gathering system-specific parameters for jobs running on BG/Q systems, a module for gathering Lustre striping data for files on Lustre file systems, and a module for instrumenting stdio (i.e., stream I/O functions like `fopen()`, `fread()`, etc).

Starting in version 3.1.3, Darshan also allows for full tracing of application I/O workloads using the newly developed Darshan eXtended Tracing (DxT) instrumentation module. This module can be selectively enabled at runtime to provide high-fidelity traces of an application's I/O workload, as opposed to the coarse-grained I/O summary data that Darshan has traditionally provided. Currently, DxT only traces at the POSIX and MPI-IO layers. Initial performance results demonstrate the low overhead of DxT tracing, offering comparable performance to Darshan's traditional coarse-grained instrumentation methods.

This document provides generic installation instructions, but "recipes" for several common HPC systems are provided at the end of the document as well.

More information about Darshan can be found at the Darshan web site.

Requirements

- MPI C compiler
- zlib development headers and library

Compilation

Configure and build example

```
tar -xvzf darshan-<version-number>.tar.gz
cd darshan-<version-number>/darshan-runtime
./configure --with-mem-align=8 --with-log-path=/darshan-logs --with-jobid-env=PBS_JOBID CC= ↩
    mpicc
make
make install
```

EXPLANATION OF CONFIGURE ARGUMENTS:

- `--with-mem-align=` (mandatory): This value is system-dependent and will be used by Darshan to determine if the buffer for a read or write operation is aligned in memory.
- `--with-jobid-env=` (mandatory): this specifies the environment variable that Darshan should check to determine the jobid of a job. Common values are `PBS_JOBID` or `COBALT_JOBID`. If you are not using a scheduler (or your scheduler does not advertise the job ID) then you can specify `NONE` here. Darshan will fall back to using the pid of the rank 0 process if the specified environment variable is not set.
- `--with-log-path=` (this, or `--with-log-path-by-env`, is mandatory): This specifies the parent directory for the directory tree where darshan logs will be placed.

- `--with-log-path-by-env=`: specifies an environment variable to use to determine the log path at run time.
- `--with-log-hints=`: specifies hints to use when writing the Darshan log file. See `./configure --help` for details.
- `--with-mod-mem=`: specifies the maximum amount of memory (in MiB) that active Darshan instrumentation modules can collectively consume.
- `--with-zlib=`: specifies an alternate location for the zlib development header and library.
- `CC=`: specifies the MPI C compiler to use for compilation.
- `--enable-mmap-logs`: enables the use of Darshan's mmap log file mechanism.
- `--disable-cuserid`: disables use of cuserid() at runtime.
- `--disable-ld-preload`: disables building of the Darshan LD_PRELOAD library
- `--disable-bgq-mod`: disables building of the BG/Q module (default checks and only builds if BG/Q environment detected).
- `--enable-group-readable-logs`: sets darshan log file permissions to allow group read access.
- `--enable-HDF5-pre-1.10`: enables the Darshan HDF5 instrumentation module, with support for HDF5 versions prior to 1.10
- `--enable-HDF5-post-1.10`: enables the Darshan HDF5 instrumentation module, with support for HDF5 versions 1.10 or higher

Cross compilation

On some systems (notably the IBM Blue Gene series), the login nodes do not have the same architecture or runtime environment as the compute nodes. In this case, you must configure darshan-runtime to be built using a cross compiler. The following configure arguments show an example for the BG/P system:

```
--host=powerpc-bgp-linux CC=/bgsys/drivers/ppcfloor/comm/default/bin/mpicc
```

Environment preparation

Once darshan-runtime has been installed, you must prepare a location in which to store the Darshan log files and configure an instrumentation method.

Log directory

This step can be safely skipped if you configured darshan-runtime using the `--with-log-path-by-env` option. A more typical configuration uses a static directory hierarchy for Darshan log files.

The `darshan-mk-log-dirs.pl` utility will configure the path specified at configure time to include subdirectories organized by year, month, and day in which log files will be placed. The deepest subdirectories will have sticky permissions to enable multiple users to write to the same directory. If the log directory is shared system-wide across many users then the following script should be run as root.

```
darshan-mk-log-dirs.pl
```

A note about log directory permissions
All log files written by darshan have permissions set to only allow read access by the owner of the file. You can modify this behavior, however, by specifying the --enable-group-readable-logs option at configure time. One notable deployment scenario would be to configure Darshan and the log directories to allow all logs to be readable by both the end user and a Darshan administrators group. This can be done with the following steps:

- set the --enable-group-readable-logs option at configure time
- create the log directories with darshan-mk-log-dirs.pl
- recursively set the group ownership of the log directories to the Darshan administrators group
- recursively set the setgid bit on the log directories

Instrumentation method

The instrumentation method to use depends on whether the executables produced by your MPI compiler are statically or dynamically linked. If you are unsure, you can check by running `ldd <executable_name>` on an example executable. Dynamically-linked executables will produce a list of shared libraries when this command is executed.

Most MPI compilers allow you to toggle dynamic or static linking via options such as `-dynamic` or `-static`. Please check your MPI compiler man page for details if you intend to force one mode or the other.

Instrumenting statically-linked applications

Statically linked executables must be instrumented at compile time. The simplest methods to do this are to either generate a customized MPI compiler script (e.g. `mpicc`) that includes the link options and libraries needed by Darshan, or to use existing profiling configuration hooks for existing MPI compiler scripts. Once this is done, Darshan instrumentation is transparent; you simply compile applications using the darshan-enabled MPI compiler scripts.

Using a profile configuration

The MPICH MPI implementation supports the specification of a profiling library configuration that can be used to insert Darshan instrumentation without modifying the existing MPI compiler script. Example profiling configuration files are installed with Darshan 2.3.1 and later. You can enable a profiling configuration using environment variables or command line arguments to the compiler scripts:

Example for MPICH 3.1.1 or newer:

```
export MPICC_PROFILE=$DARSHAN_PREFIX/share/mpi-profile/darshan-cc
export MPICXX_PROFILE=$DARSHAN_PREFIX/share/mpi-profile/darshan-cxx
export MPIFORT_PROFILE=$DARSHAN_PREFIX/share/mpi-profile/darshan-f
```

Example for MPICH 3.1 or earlier:

```
export MPICC_PROFILE=$DARSHAN_PREFIX/share/mpi-profile/darshan-cc
export MPICXX_PROFILE=$DARSHAN_PREFIX/share/mpi-profile/darshan-cxx
export MPIF77_PROFILE=$DARSHAN_PREFIX/share/mpi-profile/darshan-f
export MPIF90_PROFILE=$DARSHAN_PREFIX/share/mpi-profile/darshan-f
```

Examples for command line use:

```
mpicc -profile=$DARSHAN_PREFIX/share/mpi-profile/darshan-c <args>
mpicxx -profile=$DARSHAN_PREFIX/share/mpi-profile/darshan-cxx <args>
mpif77 -profile=$DARSHAN_PREFIX/share/mpi-profile/darshan-f <args>
mpif90 -profile=$DARSHAN_PREFIX/share/mpi-profile/darshan-f <args>
```

Using customized compiler wrapper scripts

For MPICH-based MPI libraries, such as MPICH1, MPICH2, or MVAPICH, custom wrapper scripts can be generated to automatically include Darshan instrumentation. The following example illustrates how to produce wrappers for C, C++, and Fortran compilers:

```
darshan-gen-cc.pl `which mpicc` --output mpicc.darshan
darshan-gen-cxx.pl `which mpicxx` --output mpicxx.darshan
darshan-gen-fortran.pl `which mpif77` --output mpif77.darshan
darshan-gen-fortran.pl `which mpif90` --output mpif90.darshan
```

Other configurations

Please see the Cray recipe in this document for instructions on instrumenting statically-linked applications on that platform.

For other MPI Libraries you must manually modify the MPI compiler scripts to add the necessary link options and libraries. Please see the `darshan-gen-*` scripts for examples or contact the Darshan users mailing list for help.

Instrumenting dynamically-linked applications

For dynamically-linked executables, darshan relies on the `LD_PRELOAD` environment variable to insert instrumentation at run time. The executables should be compiled using the normal, unmodified MPI compiler.

To use this mechanism, set the `LD_PRELOAD` environment variable to the full path to the Darshan shared library. The preferred method of inserting Darshan instrumentation in this case is to set the `LD_PRELOAD` variable specifically for the application of interest. Typically this is possible using command line arguments offered by the `mpirun` or `mpiexec` scripts or by the job scheduler:

```
mpiexec -n 4 -env LD_PRELOAD /home/carns/darshan-install/lib/libdarshan.so mpi-io-test
```

```
srun -n 4 --export=LD_PRELOAD=/home/carns/darshan-install/lib/libdarshan.so mpi-io-test
```

For sequential programs, the following will set LD_PRELOAD for process duration only:

```
env LD_PRELOAD=/home/carns/darshan-install/lib/libdarshan.so mpi-io-test
```

Other environments may have other specific options for controlling this behavior. Please check your local site documentation for details.

It is also possible to just export LD_PRELOAD as follows, but it is recommended against doing that to prevent Darshan and MPI symbols from being pulled into unrelated binaries:

```
export LD_PRELOAD=/home/carns/darshan-install/lib/libdarshan.so
```

Note
For SGI systems running the MPT environment, it may be necessary to set the `MPI_SHEPHERD` environment variable equal to `true` to avoid deadlock when preloading the Darshan shared library.

Instrumenting dynamically-linked Fortran applications

Please follow the general steps outlined in the previous section. For Fortran applications compiled with MPICH you may have to take the additional step of adding `libfmpich.so` to your `LD_PRELOAD` environment variable. For example:

```
export LD_PRELOAD=/path/to/mpi/used/by/executable/lib/libfmpich.so:/home/carns/darshan- ↩
    install/lib/libdarshan.so
```

Note
The full path to the libfmpich.so library can be omitted if the rpath variable points to the correct path. Be careful to check the rpath of the darshan library and the executable before using this configuration, however. They may provide conflicting paths. Ideally the rpath to the MPI library would **not** be set by the darshan library, but would instead be specified exclusively by the executable itself. You can check the rpath of the darshan library by running `objdump -x /home/carns/darshan-install/lib/libdarshan.so |grep RPATH`.

Darshan installation recipes

The following recipes provide examples for prominent HPC systems. These are intended to be used as a starting point. You will most likely have to adjust paths and options to reflect the specifics of your system.

IBM Blue Gene (BG/P or BG/Q)

IBM Blue Gene systems produces static executables by default, uses a different architecture for login and compute nodes, and uses an MPI environment based on MPICH.

The following example shows how to configure Darshan on a BG/P system:

```
./configure --with-mem-align=16 \
 --with-log-path=/home/carns/working/darshan/releases/logs \
 --prefix=/home/carns/working/darshan/install --with-jobid-env=COBALT_JOBID \
 --with-zlib=/soft/apps/zlib-1.2.3/ \
 --host=powerpc-bgp-linux CC=/bgsys/drivers/ppcfloor/comm/default/bin/mpicc
```

Rationale
The memory alignment is set to 16 not because that is the proper alignment for the BG/P CPU architecture, but because that is the optimal alignment for the network transport used between compute nodes and I/O nodes in the system. The jobid environment variable is set to `COBALT_JOBID` in this case for use with the Cobalt scheduler, but other BG/P systems may use different schedulers. The `--with-zlib` argument is used to point to a version of zlib that has been compiled for use on the compute nodes rather than the login node. The `--host` argument is used to force cross-compilation of Darshan. The `CC` variable is set to point to a stock MPI compiler.

Once Darshan has been installed, you can use one of the static instrumentation methods described earlier in this document. If you use the profiling configuration file method, then please note that the Darshan installation includes profiling configuration files that have been adapted specifically for the Blue Gene environment. Set the following environment variables to enable them, and then use your normal compiler scripts. This method is compatible with both GNU and IBM compilers.

Blue Gene profiling configuration example:

```
export MPICC_PROFILE=$DARSHAN_PREFIX/share/mpi-profile/darshan-bg-cc
export MPICXX_PROFILE=$DARSHAN_PREFIX/share/mpi-profile/darshan-bg-cxx
export MPIF77_PROFILE=$DARSHAN_PREFIX/share/mpi-profile/darshan-bg-f
export MPIF90_PROFILE=$DARSHAN_PREFIX/share/mpi-profile/darshan-bg-f
```

Cray platforms (XE, XC, or similar)

The Cray programming environment produces static executables by default, which means that Darshan instrumentation must be inserted at compile time. This can be accomplished by loading a software module that sets appropriate environment variables to modify the Cray compiler script link behavior. This section describes how to compile and install Darshan, as well as how to use a software module to enable and disable Darshan instrumentation.

Building and installing Darshan

Please set your environment to use the GNU programming environment before configuring or compiling Darshan. Although Darshan can be built with a variety of compilers, the GNU compilers are recommended because it will produce a Darshan library that is interoperable with the widest range of compilers and linkers. On most Cray systems you can enable the GNU programming environment with a command similar to "module swap PrgEnv-pgi PrgEnv-gnu". Please see your site documentation for information about how to switch programming environments.

The following example shows how to configure and build Darshan on a Cray system using the GNU programming environment. Adjust the --with-log-path and --prefix arguments to point to the desired log file path and installation path, respectively.

```
module swap PrgEnv-pgi PrgEnv-gnu
./configure --with-mem-align=8 \
 --with-log-path=/shared-file-system/darshan-logs \
 --prefix=/soft/darshan-2.2.3 \
 --with-jobid-env=PBS_JOBID --disable-cuserid CC=cc
make install
module swap PrgEnv-gnu PrgEnv-pgi
```

Rationale

The job ID is set to `PBS_JOBID` for use with a Torque or PBS based scheduler. The `CC` variable is configured to point the standard MPI compiler.

The --disable-cuserid argument is used to prevent Darshan from attempting to use the cuserid() function to retrieve the user name associated with a job. Darshan automatically falls back to other methods if this function fails, but on some Cray environments (notably the Beagle XE6 system as of March 2012) the cuserid() call triggers a segmentation fault. With this option set, Darshan will typically use the LOGNAME environment variable to determine a userid.

As in any Darshan installation, the darshan-mk-log-dirs.pl script can then be used to create the appropriate directory hierarchy for storing Darshan log files in the --with-log-path directory.

Note that Darshan is not currently capable of detecting the stripe size (and therefore the Darshan FILE_ALIGNMENT value) on Lustre file systems. If a Lustre file system is detected, then Darshan assumes an optimal file alignment of 1 MiB.

Enabling Darshan instrumentation

Darshan will automatically install example software module files in the following locations (depending on how you specified the --prefix option in the previous section):

```
/soft/darshan-2.2.3/share/craype-1.x/modulefiles/darshan
/soft/darshan-2.2.3/share/craype-2.x/modulefiles/darshan
```

Select the one that is appropriate for your Cray programming environment (see the version number of the craype module in `module list`).

If you are using the Cray Programming Environment version 1.x, then you must modify the corresponding modulefile before using it. Please see the comments at the end of the file and choose an environment variable method that is appropriate for your system. If this is not done, then the compiler may fail to link some applications when the Darshan module is loaded.

If you are using the Cray Programming Environment version 2.x then you can likely use the modulefile as is. Note that it pulls most of its configuration from the lib/pkgconfig/darshan-runtime.pc file installed with Darshan.

The modulefile that you select can be copied to a system location, or the install location can be added to your local module path with the following command:

```
module use /soft/darshan-2.2.3/share/craype-<VERSION>/modulefiles
```

From this point, Darshan instrumenation can be enabled for all future application compilations by running "module load darshan".

Linux clusters using Intel MPI

Most Intel MPI installations produce dynamic executables by default. To configure Darshan in this environment you can use the following example:

```
./configure --with-mem-align=8 --with-log-path=/darshan-logs --with-jobid-env=PBS_JOBID CC= ↩
    mpicc
```

Rationale
There is nothing unusual in this configuration except that you should use the underlying GNU compilers rather than the Intel ICC compilers to compile Darshan itself.

You can use the `LD_PRELOAD` method described earlier in this document to instrument executables compiled with the Intel MPI compiler scripts. This method has been briefly tested using both GNU and Intel compilers.

Caveat
Darshan is only known to work with C and C++ executables generated by the Intel MPI suite in versions prior to the 2017 version — Darshan will not produce instrumentation for Fortran executables in these earlier versions (pre-2017). For more details on this issue please check this Intel forum discussion:
http://software.intel.com/en-us/forums/showthread.php?t=103447&o=a&s=lr

Linux clusters using MPICH

Follow the generic instructions provided at the top of this document. For MPICH versions 3.1 and later, MPICH uses shared libraries by default, so you may need to consider the dynamic linking instrumentation approach.

The static linking method can be used if MPICH is configured to use static linking by default, or if you are using a version prior to 3.1. The only modification is to make sure that the `CC` used for compilation is based on a GNU compiler. Once Darshan has been installed, it should be capable of instrumenting executables built with GNU, Intel, and PGI compilers.

Note
Darshan is not capable of instrumenting Fortran applications build with MPICH versions 3.1.1, 3.1.2, or 3.1.3 due to a library symbol name compatibility issue. Consider using a newer version of MPICH if you wish to instrument Fortran applications. Please see http://trac.mpich.org/projects/mpich/ticket/2209 for more details.

Note
MPICH versions 3.1, 3.1.1, 3.1.2, and 3.1.3 may produce link-time errors when building static executables (i.e. using the -static option) if MPICH is built with shared library support. Please see http://trac.mpich.org/projects/mpich/ticket/2190 for more details. The workaround if you wish to use static linking is to configure MPICH with `--enable-shared=no --enable-static=yes` to force it to use static MPI libraries with correct dependencies.

Linux clusters using Open MPI

Follow the generic instructions provided at the top of this document for compilation, and make sure that the `CC` used for compilation is based on a GNU compiler.

Open MPI typically produces dynamically linked executables by default, which means that you should use the `LD_PRELOAD` method to instrument executables that have been built with Open MPI. Darshan is only compatible with Open MPI 1.6.4 and newer. For more details on why Darshan is not compatible with older versions of Open MPI, please refer to the following mailing list discussion:

http://www.open-mpi.org/community/lists/devel/2013/01/11907.php

Upgrading to Darshan 3.x from 2.x

Beginning with Darshan 3.0.0, Darshan has been rewritten to modularize its runtime environment and log file format to simplify the addition of new I/O characterization data. The process of compiling and installing the Darshan 3.x source code should essentially be identical to this process on Darshan 2.x. Therefore, the installation recipes given in the previous section should work irrespective of the Darshan version being used. Similarly, the manner in which Darshan is used should be the same across versions — the sections in this document regarding Darshan environment preparation, instrumenting statically linked applications and dynamically linked applications, and using runtime environment variables are equally applicable to both versions.

However, we do provide some suggestions and expectations for system administrators to keep in mind when upgrading to Darshan 3.x:

- Darshan 2.x and Darshan 3.x produce incompatible log file formats
  - log files named *.darshan.gz or *.darshan.bz2: Darshan 2.x format
  - log files named *.darshan: Darshan 3.x format
    - a field in the log file header indicates underlying compression method in 3.x logs
  - There is currently no tool for converting 2.x logs into the 3.x log format.
  - The `darshan-logutils` library will provide error messages to indicate whether a given log file is incompatible with the correspnonding library version.
- We encourage administrators to use the same log file directory for version 3.x as had been used for version 2.x.
  - Within this directory, the determination on which set of log utilities (version 2.x or version 3.x) to use can be based on the file extension for a given log (as explained above).

Runtime environment variables

The Darshan library honors the following environment variables to modify behavior at runtime:

- DARSHAN_DISABLE: disables Darshan instrumentation
- DARSHAN_INTERNAL_TIMING: enables internal instrumentation that will print the time required to startup and shutdown Darshan to stderr at run time.
- DARSHAN_LOGHINTS: specifies the MPI-IO hints to use when storing the Darshan output file. The format is a semicolon-delimited list of key=value pairs, for example: hint1=value1;hint2=value2
- DARSHAN_MEMALIGN: specifies a value for system memory alignment
- DARSHAN_JOBID: specifies the name of the environment variable to use for the job identifier, such as PBS_JOBID
- DARSHAN_DISABLE_SHARED_REDUCTION: disables the step in Darshan aggregation in which files that were accessed by all ranks are collapsed into a single cumulative file record at rank 0. This option retains more per-process information at the expense of creating larger log files. Note that it is up to individual instrumentation module implementations whether this environment variable is actually honored.
- DARSHAN_LOGPATH: specifies the path to write Darshan log files to. Note that this directory needs to be formatted using the darshan-mk-log-dirs script.
- DARSHAN_LOGFILE: specifies the path (directory + Darshan log file name) to write the output Darshan log to. This overrides the default Darshan behavior of automatically generating a log file name and adding it to a log file directory formatted using darshan-mk-log-dirs script.
- DARSHAN_MODMEM: specifies the maximum amount of memory (in MiB) Darshan instrumentation modules can collectively consume at runtime (if not specified, Darshan uses a default quota of 2 MiB).

- DARSHAN_MMAP_LOGPATH: if Darshan's mmap log file mechanism is enabled, this variable specifies what path the mmap log files should be stored in (if not specified, log files will be stored in `/tmp`).
- DARSHAN_EXCLUDE_DIRS: specifies a list of comma-separated paths that Darshan will not instrument at runtime (in addition to Darshan's default blacklist)
- DXT_ENABLE_IO_TRACE: setting this environment variable enables the DXT (Darshan eXtended Tracing) modules at runtime. Users can specify a numeric value for this variable to set the number of MiB to use for tracing per process; if no value is specified, Darshan will use a default value of 4 MiB.

Debugging

No log file

In cases where Darshan is not generating a log file for an application, some common things to check are:

- Check stderr to ensure Darshan isn't indicating any internal errors (e.g., invalid log file path)

For statically linked executables:

- Ensure that Darshan symbols are present in the underlying executable by running `nm` on it:

```
> nm test | grep darshan
0000000000772260 b darshan_core
0000000000404440 t darshan_core_cleanup
00000000004049b0 T darshan_core_initialize
000000000076b660 d darshan_core_mutex
00000000004070a0 T darshan_core_register_module
```

- Make sure the application executable is statically linked:
  - In general, we encourage the use of purely statically linked executables when using the static instrumentation method given in Section 5
  - If purely static executables are not an option, we encourage users to use the LD_PRELOAD method of instrumentation given in Section 6
  - Statically linked executables are the default on Cray platforms and the IBM BG platforms; statically linked executables can be explicitly requested using the `-static` compile option to most compilers
  - You can verify that an executable is purely statically linked by using the `file` command:

```
> file mpi-io-test
mpi-io-test: ELF 64-bit LSB  executable, x86-64, version 1 (GNU/Linux), statically linked,  ↩
    for GNU/Linux 2.6.24, BuildID[sha1]=9893e599e7a560159ccf547b4c4ba5671f65ba32, not  ↩
    stripped
```

- Ensure that the linker is correctly linking in Darshan's runtime libraries:
  - A common mistake is to explicitly link in the underlying MPI libraries (e.g., `-lmpich` or `-lmpichf90`) in the link command, which can interfere with Darshan's instrumentation
    - These libraries are usually linked in automatically by the compiler
    - MPICH's `mpicc` comipler's `-show` flag can be used to examine the invoked link command, for instance
  - The linker's `-y` option can be used to verify that Darshan is properly intercepting MPI_Init function (e.g. by setting `CFLAGS='-Wl,-yMPI_Init'`), which it uses to initialize its runtime structures

```
/usr/common/software/darshan/3.0.0-pre3/lib/libdarshan.a(darshan-core-init-finalize.o):  ↩
    definition of MPI_Init
```