SOURCEFORGE.NET®

mpiP: Lightweight, Scalable MPI Profiling

Version 3.4.1
13 March 2014

Jeffrey Vetter
jsvetter@users.sourceforge.net

Chris Chambreau
chcham@users.sourceforge.net

Contents

Introduction

mpiP is a lightweight profiling library for MPI applications. Because it only collects statistical information about MPI functions, mpiP generates considerably less overhead and much less data than tracing tools. All the information captured by mpiP is task-local. It only uses communication during report generation, typically at the end of the experiment, to merge results from all of the tasks into one output file.

We have tested mpiP on a variety of C/C++/Fortran applications from 2 to 262144 processes, including a 262144-process run on the LLNL Sequoia BG/Q system.

Please send your comments, questions, and ideas for enhancements to mpip-help@lists.sourceforge.net. To receive mail regarding new mpiP releases, please subscribe to mpip-announce@lists.sourceforge.net (send e-mail with body "subscribe" to mpip-announce-request@lists.sourceforge.net). Please also consider subscribing to mpip-users@lists.sourceforge.net to contribute and receive mpiP use and status information.

To learn more about performance analysis with mpiP, see Vetter, J.S. and M.O. McCracken, "Statistical Scalability Analysis of Communication Operations in Distributed Applications," Proc. ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP), 2001.

Downloading

You may download the current version of mpiP from http://sourceforge.net/projects/mpip.

Contributing

We are constantly improving mpiP. Bug fixes and ports to new platforms are always welcome. Many thanks to the following contributors (chronological order):

- Michael McCracken (UCSD)
- Curt Janssen (Sandia National Laboratories)
- Mike Campbell (UIUC)
- Jim Brandt (Sandia National Laboratories)
- Philip Roth (Oak Ridge National Laboratory)
- Tushar Mohan (SiCortex)
- Philip Mucci (SiCortex)

- Karl Schulz (Texas Advanced Computing Center)

New Features with Release 3.4.1

Release v3.4.1 addresses the following issue:

- Added de-activation of shared object source lookup when libbfd is not available.

Release v3.4 addresses the following issues:

- Compatibility with MPI-3.
- Histogram reporting for Point-to-point (-p) and Collective (-y) operation message sizes and communicators.
- Added a low-memory-use concise report format, with the ability to set the default report format and specify report formats at run time.
- Supports MPI call reporting (no call sites) with stack depth (-k) of 0.
- Configure can disable SO lookup functionality.

Release v3.3 addresses the following issues:

- Support for shared object source lookup with libbfd.
- Improved configuration process for recent versions of binutils and Cray XE6.
- Added "-z" MPIP run time flag to suppress report generation at Finalize.
- Corrected number of stack frames available when using glibc backtrace.

Release v3.2.1 addresses the following issue:

- Improved support for SLURM run-time instrumentation.

Release v3.2 addresses the following issues:

- Support for MPI RMA functions.
- Support for glibc backtrace.
- Default to MPI_Wtime if platform-specific timers are not found.

Release v3.1.2 addresses the following issues:

- Better MPI support for Init_thread, Testany, Testsome, Waitany, and Waitsome.
- Improved support for MIPS64-Linux.
- Added option to configure for generating weak Fortran symbols in the case of multiple Fortran mangling schemes in the application object files (--enable-fortranweak).
- Addressed various outstanding issues (see ChangeLog for more details).

Release v3.1.1 addresses the following issues:

- Revert to gettimeofday as default Linux timer.
- MIPS64-Linux stack walking support.
- Catamount dclock timer support.
- Greater install flexibility:
  - 'install' target only installs lib and doc files.
  - 'install-api', 'install-bin', 'install-all' targets provide additional install functionality.
  - New 'uninstall' target.

For more information, please see the ChangeLog in the distribution.

Top

Using mpiP

Using mpiP is very simple. Because it gathers MPI information through the MPI profiling layer, mpiP is a link-time library. That is, you don't have to recompile your application to use mpiP. Note that you might have to recompile to include the '-g' option. This is important if you want mpiP to decode the PC to a source code filename and line number automatically. mpiP will work without `-g`, but mileage may vary.

To compile a simple program on an LLNL x86_64-linux system where libunwind is installed, add the following libraries to your link command:

```
-L${mpiP_root}/lib -lmpiP -lm -lbfd -liberty -lunwind
```

For example, the new mpiP link command becomes

```
$ mpicc -g 1-hot-potato.o -o 1-hot-potato.exe -L${mpiP_root}/lib -lmpiP -lm -lbfd -liberty -lunwind
```

from

```
$ mpicc -g 1-hot-potato.o -o 1-hot-potato.exe
```

Make sure the mpiP library appears before the MPI library on your link line. The libraries (`-lbfd -liberty`) provide support for decoding the symbol information; they are part of GNU binutils.

Run your application. You can verify that mpiP is working by identifying the header and trailer in standard out.

```
mpiP:
mpiP: mpiP: mpiP V3.2.0 (Build Mar 10 2010/13:27:39)
mpiP: Direct questions and errors to mpip-help@lists.sourceforge.net
mpiP:
mpiP:
mpiP: Storing mpiP output in [./1-hot-potato.exe.2.27872.1.mpiP].
mpiP:
```

By default, the output file is written to the current directory of the application. mpiP files are always much smaller than trace files, so writing them to this directory is safe.

Supported Platforms

mpiP has been tested on several Linux, AIX, UNICOS and IBM BG systems. Please contact us with bug reports or questions regarding these platforms. The following table indicates platforms where mpiP was succesfully run and any requirements for that platform.

Platform	OS	Compiler	MPI	binutils	Requirements
x86_64-Linux	2.6.18 CHAOS Kernel	Intel 9.1 PGI 7.0	MVAPICH 0.9.7	2.20.51	Example configure command: ./configure LDFLAGS=-L/usr/lib64 LIBS="-lbfd -liberty" --enable-collective-report-default --enable-demangling=GNU --with-cc=mpicc --with-cxx=mpiCC --with-f77=mpif77
IBM BG/Q	Driver V1R2M0	IBM XL 12.1		2.21.1	Source code lookup support requires zlib. Example configure command: CFLAGS=-I/bgsys/drivers/ppcfloor/toolchain/gnu/build-powerpc64-bgq-linux/binutils-2.21.1-build/bfd -I/bgsys/drivers/ppcfloor/toolchain/gnu/gcc-4.4.6/include -I/bgsys/drivers/ppcfloor/toolchain/gnu/gdb-7.1/include LIBS=-L/bgsys/drivers/ppcfloor/toolchain/gnu/build-powerpc64-bgq-linux/binutils-2.21.1-build/bfd -L/g/g0/chcham/ToolTesting/mpiP/bgqos_0/mpiP-3.3/libz/zlib-1.2.6 CC=mpixlc_r CXX=mpixlcxx_r F77=mpixlf77_r ./configure --enable-getarg
Cray XT3/XT4	Catamount 1.4.32	PGI 6.1-4 C/C/Fortran, from Cray PrgEnv-pgi module version 1.4.32	Cray XT3 Message Passing Toolkit (MPT) version 1.4.32	Binutils 2.17, built for x86_64-unknown-linux-gnu	--with-cc=cc --with-cxx=CC --with-f77=ftn --with-libs="-lpgf90 -lpgf90_rpm1 -lpgf902 -lpgf90rtl -lpgftnrtl" --build=x86_64-unknown-linux-gnu --host=x86_64-cray-catamount --target=x86_64-cray-catamount --enable-getarg \\ --with-wtime \\ --with-binutils-dir=<path to binutils-2.17 installation>
Cray X1E	UNICOS/mp 3.1.16	Cray Standard C 5.5.0.5, Cray Fortran 5.5.0.5	Cray Message Passing Toolkit (MPT) 2.4.0.7	from Cray Open Software module 3.6	Requires libelf/libdwarf. Example configure flags: --disable-libunwind --enable-dwarf --disable-demangle --enable-getarg --with-cc=cc --with-cxx=CC --with-f77=ftn Python on an alternative system may be needed to "make wrappers.c", due to missing socket module.

Top

Configuring and Building mpiP

Configuring mpiP

Currently, mpiP requires a compatible GNU binutils installation for source lookup and demangling features. Alternatively, libelf and libdwarf can be used for source lookup. The binutils installation location may need to be specified with either the `--with-binutils-dir` option or with the `--with-include` and `--with-ldflags` configure flags. It is likely that the compilers will need to be indentified as well, with the `--with-cc`, `--with-cxx`, and `--with-f77` flags. Use CFLAGS and FFLAGS variables to specify compiler options, as in `CFLAGS="-O3" ./configure`.

There are many configuration options available. Please use ./configure --help to list all of these options. Additional description are provided for the following options:

Flag	Effect	Description
Reporting Options		
--enable-demangling=[type]	Specify demangling support.	If the GNU option is specified, demangling is applied to each symbol by default using the libiberty implementation. For the IBM option, demangling support is implemented in the library libmpiPdmg.a. Use GNU for the Intel compiler.

--disable-mpi-io	Disable MPI-I/O reporting.	Useful for generating an mpiP library without MPI I/O for MPI implementations such as Quadrics MPI that has a separate MPI I/O library.
--enable-collective-report-default	Report data is aggregated on a per-callsite basis	By default, mpiP aggregates all process data at a single process which generates the report. Enabling this feature causes mpiP report generation to default to aggregating callsite data only for each individual callsite being reported. This dramatically reduces the memory requirements for large runs of applications that make many MPI calls. See run-time flags -l and -r to modify report generation behavior.
Stack Trace Options		
--enable-stackdepth=[depth]	Specify maximum stacktrace depth (default is 8).	Stacktraces with larger than 8 levels are sometime useful for some applications.
--disable-libunwind	Do not use libunwind to generate stack traces.	Currently, libunwind seems useful on IA64-Linux and x86-Linux platforms, although it can conflict with the libunwind.a provided with the Intel compiler.
Address Lookup Options		
--enable-dwarf	Use libdwarf/libelf for source lookup.	libdwarf and libelf can be used for address-to-source translation as an alternative to binutils libbfd.
--disable-bfd	Do not use GNU binutils libbfd for source lookup.	Binutils is not always available or compatible.
Timing Options		
--with-gettimeofday	Use gettimeofday for timing.	Use the gettimeofday call for timing instead of the default platform timer.
--with-wtime	Use MPI_Wtime for timing	Use the MPI_Wtime call for timing instead of the default platform timer.
--with-clock_gettime	Use clock_gettime for timing.	Use the clock_gettime monotonic timer for timing instead of the default platform timer.
--with-dclock	Use Catamount dclock for timing.	Use the dclock timer for timing on Catamount systems instead of the default platform timer.
--enable-check-time	Enable AIX check for negative time values.	Activate IBM timing debugging code.
Fortran-related Options		
--enable-getarg	Use getarg to get Fortran command line args.	This is used on UNICOS to provide access to the command line for Fortran apllications.
--disable-fortran-xlate	Do not translate Fortran opaque objects.	Opaque object translation is not necessary on some platforms, but necessary for Fortran applications on some 64-bit platforms.
--enable-fortranweak	Generate weak symbols for additional Fortran symbol name styles.	If application objects have been created from compilers with different Fortran symbol name styles, it may be necessary to generate weak symbols to capture all MPI calls.

Build targets

Command	Effect

make	Build mpiP library or libraries for MPI profiling
make install	Install bin, include, lib, and slib (if applicable) directories. The default install directory is the mpiP source directory. The installation location can be specified with the prefix variable as in `make prefix=[install directory] install`.
make shared	Make shared object version of library for runtime insertion (Linux). Support for runtime insertion on AIX and for MPI calls made within shared objects on Linux and AIX will be provided in a future release.
make API	Make standalone API library. See mpiP-API.c and mpiP-API.h for available features.
make check	Run mpiP dejagnu tests. Requires that runtest is available.
make add_binutils_objs	For convenience, add the binutils objects to the mpiP library. The binutils installation location must have been specified during configuration.
make add_libunwind_objs	For convenience, add the libunwind objects to the mpiP library.

Example Application Link Commands

LLNL users can now use the srun-mpip, poe-mpip, and poe-mpip-cxx wrapper scripts to use mpiP without re-linking their application. AIX executables would still need to be linked with -bnoobjreorder for successful runtime address lookup. Additionally, all LLNL installations contain the appropriate binutils objects in the mpiP library, so the -lbfd, -liberty, and -lintl flags are not required. An example runtime script for mpirun is provided in the mpiP bin directory. Many of the following examples use LLNL-specific compile scripts.

OS	Compiler	Language	Example Link Command
AIX	Visual Age	C	`mpxlc -g -bnoobjreorder 1-hot-potato.c -o 1-hot-potato.exe -L/usr/local/tools/mpiP/lib -lmpiP -lbfd -liberty -lintl -lm`
		C++	`mpCC_r -g -bnoobjreorder 4-demangle.C -o 4-demangle.exe -L/usr/local/tools/mpiP/lib -lmpiPdmg -lbfd -liberty -lintl -lm`
		Fortran	`mpxlf -g -bnoobjreorder sweep-ops.f -o sweep-ops.exe -L/usr/local/tools/mpiP/lib -lmpiP -lbfd -liberty -lintl -lm`
Linux	Intel	C	`mpiicc -g 1-hot-potato.c -o 1-hot-potato.exe -L/usr/local/tools/mpiP/lib -lmpiP -lbfd -liberty -lm -lmpio`
		C++	`mpiicc -g 4-demangle.C -o 4-demangle.exe -L/usr/local/tools/mpiP/lib -lmpiP -lbfd -liberty -lm -lmpio`
		Fortran	`mpiifc -g sweep-ops.f -o sweep-ops.exe -L/usr/local/tools/mpiP/lib -lmpiP -lbfd -liberty -lm -lmpio`
	PGI	C	`mpipgcc -g 1-hot-potato.c -o 1-hot-potato.exe -L/usr/local/tools/mpiP/lib -lmpiP -lbfd -liberty -lm -lmpio`
		C++	`mpipgCC -g 4-demangle.C -o 4-demangle.exe -L/usr/local/tools/mpiP/lib -lmpiP -lbfd -liberty -lm -lmpio`
		Fortran	`mpipgf77 -g sweep-ops.f -o sweep-ops.exe -L/usr/local/tools/mpiP/lib -lmpiP -lbfd -liberty -lm -lmpio`
	GNU	C	`mpicc -g 1-hot-potato.c -o 1-hot-potato.exe -L/usr/local/tools/mpiP/lib -lmpiP -lbfd -liberty -lm`
		C++	`mpiCC -g 4-demangle.C -o 4-demangle.exe -L/usr/local/tools/mpiP/lib -lmpiP -lbfd -liberty -lm`
		Fortran	`mpif77 -g sweep-ops.f -o sweep-ops.exe -L/usr/local/tools/mpiP/lib -lmpiP -lbfd -liberty -lm`
Cray X1	Cray	C/C++/Fortran	Link with `-lmpiP -lbfd -liberty -ldwarf -lelf`
Cray XD1	GNU or PGI	C/C++/Fortran	Link with `[path_to_mpiP_install]/libmpiP.a -lbfd -liberty [mpich_libs]`

Note:

- If source lookup is failing during report generation, the script mpip-insert-src can be used from a login node to translate addresses in the mpiP report to source information.
- Source lookup for callsites may fail with certain versions of binutils. If you are running into trouble, you may want to download a recent snapshot from ftp://ftp.gnu.org/gnu/binutils/.

Top

Run-time Configuration of mpiP

mpiP has several configurable parameters that a user can set via the environment variable MPIP. Setting MPIP is done like command-line parameters: "-t 10 -k 2". Additionally, a comma can be used to delimit multiple parameters, as in "-t10,-k2". Currently, mpiP has several configurable parameters.

Option	Description	Default

`-c`	Generate concise version of report, omitting callsite process-specific detail.	
`-d`	Suppress printing of callsite detail sections.	
`-e`	Print report data using floating-point format.	
`-f dir`	Record output file in directory <dir>.	.
`-g`	Enable mpiP debug mode.	disabled
`-k n`	Sets callsite stack traceback depth to <n>.	1
`-l`	Use less memory to generate the report by using MPI collectives to generate callsite information on a callsite-by-callsite basis.	
`-n`	Do not truncate full pathname of filename in callsites.	
`-o`	Disable profiling at initialization. Application must enable profiling with MPI_Pcontrol().	
`-p`	Point-to-point histogram reporting on message size and communicator used.	
`-r`	Generate the report by aggregating data at a single task.	default
`-s n`	Set hash table size to <n>.	256
`-t x`	Set print threshold for report, where <x> is the MPI percentage of time for each callsite.	0.0
`-v`	Generates both concise and verbose report output.	
`-x exe`	Specify the full path to the executable.	
`-y`	Collective histogram reporting on message size and communicator used.	
`-z`	Suppress printing of the report at MPI_Finalize.	

For example, to set the callsite stack walking depth to 2 and the report print threshold to 10%, you simply need to define the mpiP string in your environment, as in any of the following examples:

```
$ export MPIP="-t 10.0 -k 2" (bash)
```

```
$ export MPIP=-t10.0,-k2 (bash)
```

```
$ setenv MPIP "-t 10.0 -k 2" (csh)
```

mpiP prints a message at initialization if it successfully finds this MPIP variable.

Top

mpiP Output

Here is some sample output from mpiP with an application that has 4 MPI calls. It is broken down by sections below. Here also is the experiment setup. **Note that MPIP does not capture information about ALL MPI calls**. Local calls, such as `MPI_Comm_size`, are omitted from the profiling library measurement to reduce perturbation and mpiP output.

The test code:

```
  sleeptime = 10;
  MPI_Init (&argc, &argv);
  MPI_Comm_size (comm, &nprocs);
  MPI_Comm_rank (comm, &rank);
  MPI_Barrier (comm);
  if (rank == 0)
    {
      sleep (sleeptime);        /* slacker! delaying everyone else */
    }
  MPI_Barrier (comm);
  MPI_Finalize ();
```

The code was compiled with:

```
 $ mpcc -g -DAIX 9-test-mpip-time.c -o 9-test-mpip-time.exe \
 -L.. -L/g/g2/vetter/AIX/lib -lmpiP -lbfd -liberty -lintl -lm
```

Environment variables were set as:

```
 $ export MPIP="-t 10.0"
```

The example was executed on MCR like this:

```
 $ srun -n 4 -ppdebug ./9-test-mpip-time.exe
```

This experiment produced an output file that we can now analyze:

```
 ./9-test-mpip-time.exe.4.25972.1.mpiP
```

Header information provides basic information about your performance experiment.

```
@ mpiP
@ Command : /g/g0/chcham/mpiP/devo/testing/./9-test-mpip-time.exe
@ Version : 2.8.2
```

```
@ MPIP Build date : Jan 10 2005, 15:15:47
@ Start time : 2005 01 10 16:01:32
@ Stop time : 2005 01 10 16:01:42
@ Timer Used : gettimeofday
@ MPIP env var : -t 10.0
@ Collector Rank : 0
@ Collector PID : 25972
@ Final Output Dir : .
@ MPI Task Assignment : 0 mcr88
@ MPI Task Assignment : 1 mcr88
@ MPI Task Assignment : 2 mcr89
@ MPI Task Assignment : 3 mcr89
```

This next section provides an overview of the application's time in MPI. Apptime is the wall-clock time from the end of `MPI_Init` until the beginning of `MPI_Finalize`. MPI_Time is the wall-clock time for all the MPI calls contained within Apptime. MPI% shows the ratio of this MPI_Time to Apptime. The asterisk (*) is the aggregate line for the entire application.

```
---------------------------------------------------------------------------
@--- MPI Time (seconds) ---------------------------------------------------
---------------------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0         10   0.000243     0.00
   1         10         10    99.92
   2         10         10    99.92
   3         10         10    99.92
   *         40         30    74.94
```

The callsite section identifies all the MPI callsites within the application. The first number is the callsite ID for this mpiP file. The next column shows the type of MPI call (w/o the MPI_ prefix). The name of the function that contains this MPI call is next, followed by the file name and line number. Finally, the last column shows the PC, or program counter, for that MPI callsite. Note that the default setting for callsite stack walk depth is 1. Other settings will enumerate callsites by the entire stack trace rather than the single callsite alone.

```
---------------------------------------------------------------------------
@--- Callsites: 2 ---------------------------------------------------------
---------------------------------------------------------------------------
 ID Lev File/Address        Line Parent_Funct             MPI_Call
  1   0 9-test-mpip-time.c    52 main                     Barrier
  2   0 9-test-mpip-time.c    61 main                     Barrier
```

The aggregate time section is a very quick overview of the top twenty MPI callsites that consume the most aggregate time in your application. Call identifies the type of MPI function. Site provides the callsite ID (as listed in the callsite section). Time is the aggregate time for that callsite in milliseconds. The next two columns show the ratio of that aggregate time to the total application time and to the total MPI time, respectively. The COV column indicates the variation in times of individual processes for this callsite by presenting the coefficient of variation as calculated from the individual process times. A larger value indicates more variation between the process times.

```
---------------------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
---------------------------------------------------------------------------
Call                 Site       Time    App%    MPI%     COV
Barrier                 2      3e+04   75.00  100.00    0.67
Barrier                 1      0.405    0.00    0.00    0.59
```

The next section is similar to the aggregate time section, although it reports on the top 20 callsites for total sent message sizes. For example:

```
---------------------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
---------------------------------------------------------------------------
Call                 Site      Count      Total     Avrg      MPI%
Send                    7        320   1.92e+06   6e+03     99.96
Bcast                   1         12        336      28      0.02
```

The final sections are the ad nauseum listing of the statistics for each callsite across all tasks, followed by an aggregate line (indicated by an asterisk in the Rank column). The first section is for operation time followed by the section for message sizes.

```
---------------------------------------------------------------------------
@--- Callsite Time statistics (all, milliseconds): 8 ----------------------
---------------------------------------------------------------------------
Name              Site Rank  Count      Max     Mean      Min   App%   MPI%
Barrier              1    0      1    0.107    0.107    0.107   0.00  44.03
Barrier              1    *      4    0.174    0.137    0.107   0.00   0.00

Barrier              2    0      1    0.136    0.136    0.136   0.00  55.97
Barrier              2    1      1    1e+04    1e+04    1e+04  99.92 100.00
Barrier              2    2      1    1e+04    1e+04    1e+04  99.92 100.00
Barrier              2    3      1    1e+04    1e+04    1e+04  99.92 100.00
Barrier              2    *      4    1e+04  7.5e+03    0.136  74.94 100.00
```

Remember that we configured MPIP to not print lines where MPI% was less than 10%. All aggregate lines are printed regardless of the configuration settings.

Column	Description
Name	Name of the MPI function at that callsite.

Site	Callsite ID as listed in the callsite section above.
Rank	Task rank in MPI_COMM_WORLD.
Count	Number of times this call was executed.
Max	Maximum wall-clock time for one call.
Mean	Arithmetic mean of the wall-clock time for one call.
Min	Minimum wall-clock time for one call.
App%	Ratio of time for this call to the overall application time for each task.
MPI%	Ratio of time for this call to the overall MPI time for each task.

The aggregate result for each call has the same measurement meaning; however, the statistics are gathered across all tasks and compared with the aggregate application and MPI times.

The section for sent message sizes has a similar format:

```
---------------------------------------------------------------------------
@--- Callsite Message Sent statistics (all, sent bytes) -------------------
---------------------------------------------------------------------------
Name              Site Rank   Count       Max      Mean       Min       Sum
Send                 5    0      80      6000      6000      6000   4.8e+05
Send                 5    1      80      6000      6000      6000   4.8e+05
Send                 5    2      80      6000      6000      6000   4.8e+05
Send                 5    3      80      6000      6000      6000   4.8e+05
Send                 5    *     320      6000      6000      6000   1.92e+06
```

Column	Description
Name	Name of the MPI function at that callsite.
Site	Callsite ID as listed in the callsite section above.
Rank	Task rank in MPI_COMM_WORLD.
Count	Number of times this call was executed.
Max	Maximum sent message size in bytes for one call.
Mean	Arithmetic mean of the sent message sizes in bytes for one call.
Min	Minimum sent message size in bytes for one call.
Sum	Total of all message sizes for this operation and callsite.

The format of MPI I/O report section is very similar to the sent message sizes section:

```
---------------------------------------------------------------------------
@--- Callsite I/O statistics (all, I/O bytes) -----------------------------
---------------------------------------------------------------------------
Name              Site Rank   Count       Max      Mean       Min       Sum
File_read            1    0      20        64        64        64      1280
File_read            1    1      20        64        64        64      1280
File_read            1    *      40        64        64        64      2560
```

Report Viewers

- The Tool Gear project has a Qt mpiP viewer. LLNL users can run this as mpipview.

Top

Controlling the Scope of mpiP Profiling in your Application

In mpiP, you can limit the scope of profiling measurements to specific regions of your code using the `MPI_Pcontrol(int level)` subroutine. A value of zero disables mpiP profiling, while any nonzero value enables profiling. To disable profiling initially at MPI_Init, use the `-o` configuration option. mpiP will only record information about MPI commands encountered between activation and deactivation. There is no limit to the number to times that an application can activate profiling during execution.

For example, in your application you can capture the MPI activity for timestep 5 only using Pcontrol. Remember to set the mpiP environment variable to include `-o` when using this feature.

```
for(i=1; i < 10; i++)
{
  switch(i)
  {
    case 5:
      MPI_Pcontrol(1);
      break;
    case 6:
      MPI_Pcontrol(0);
      break;
    default:
      break;
```

```
  }
  /* ... compute and communicate for one timestep ... */
}
```

Arbitrary Report Generation

You can also generate arbitrary reports by making calls to `MPI_Pcontrol()` with an argument of 3 or 4 (see table below). The first report generated will have the default report filename. Subsequent report files will have an index number included, such as `sweep3d.mpi.4.7371.1.mpiP`, `sweep3d.mpi.4.7371.2.mpiP`, etc. The final report will still be generated during MPI_Finalize. **NOTE:** In the current release, callsite IDs will not be consistent between reports. Comparison of callsite data between reports must be done by source location and callstack.

MPI_Pcontrol features should be fully functional for C/C++ as well as Fortran.

Pcontrol Argument	Behavior
0	Disable profiling.
1	Enable Profiling.
2	Reset all callsite data.
3	Generate verbose report.
4	Generate concise report.

If you want to generate individual reports each time a section of code is executed, but don't want the profile data to accumulate, you can specify code to reset the profile data, profile, and then generate reports. For example:

```
for(i=1; i < 10; i++)
{
  switch(i)
  {
    case 5:
      MPI_Pcontrol(2); // make sure profile data is reset
      MPI_Pcontrol(1); // enable profiling
      break;
    case 6:
      MPI_Pcontrol(3); // generate verbose report
      MPI_Pcontrol(4); // generate concise report
      MPI_Pcontrol(0); // disable profiling
      break;
    default:
      break;
  }
  /* ... compute and communicate for one timestep ... */
}
```

Top

Caveats

- If mpiP has problems with the source code translation, you might be able to decode the program counters on LLNL systems with some of the following techniques. You can use instmap, addr2line, or look at the assembler code itself.
- Compiler transformations like loop unrolling can sometimes make one source code line appear as many different PCs. You can verify this by looking at the assembler. In my experience, both instmap and addr2line do a pretty good job of mapping these transformed PCs into a file name and line number.
  - instmap—an IBM utility
  - addr2line—a gnu tool
  - look at the assembler listing, or with GNU's objdump (`-d -S`)
  - use Totalview or gdb to translate the PC
- There are known incompatibilities with certain binutils versions and recent versions of the IBM compilers. As of this release, a fix has not been incorporated into binutils, however, using the `-bnoobjreorder` option is a valid work-around.
- In one case, we encountered problems on IBM machines with source lookup of 64-bit Fortran applications. It appears that an incorrect compiler configuration file was being used, incorrectly matching debugging information and PC values. We addressed this by using the link flag -bpT:0x100000000.
- Issues when stack walking optimized applications:
  - Applications compiled with gcc may return incorrect parent functions; however, the file and line number information may be correct.
  - Applications compiled with the Intel compiler may not be able to identify parent stack frames.
- If you are calling MPI functions from within dynamically loaded objects, you may need to recompile the library as a shared object.
- We have encountered occaisional negative report values on Linux and AIX systems. We will continue to investigate this issue, but it is possible that this behavior may be experienced with mpiP.

Top

MPI Routines Profiled with mpiP

MPI_Allgather
MPI_Allgatherv
MPI_Allreduce
MPI_Alltoall
MPI_Alltoallv
MPI_Attr_delete
MPI_Attr_get
MPI_Attr_put
MPI_Barrier
MPI_Bcast
MPI_Bsend
MPI_Bsend_init
MPI_Buffer_attach
MPI_Buffer_detach
MPI_Cancel
MPI_Cart_coords
MPI_Cart_create
MPI_Cart_get
MPI_Cart_map
MPI_Cart_rank
MPI_Cart_shift
MPI_Cart_sub
MPI_Cartdim_get
MPI_Comm_create
MPI_Comm_dup
MPI_Comm_group
MPI_Comm_remote_group
MPI_Comm_remote_size
MPI_Comm_split
MPI_Comm_test_inter
MPI_Dims_create
MPI_Error_class
MPI_File_close
MPI_File_open
MPI_File_preallocate
MPI_File_read
MPI_File_read_all
MPI_File_read_at
MPI_File_seek
MPI_File_set_view
MPI_File_write
MPI_File_write_all
MPI_File_write_at
MPI_Gather
MPI_Gatherv
MPI_Graph_create
MPI_Graph_get
MPI_Graph_map
MPI_Graph_neighbors
MPI_Graph_neighbors_count
MPI_Graphdims_get
MPI_Group_compare
MPI_Group_difference
MPI_Group_excl
MPI_Group_free
MPI_Group_incl
MPI_Group_intersection
MPI_Group_translate_ranks
MPI_Group_union
MPI_Ibsend
MPI_Intercomm_create
MPI_Intercomm_merge
MPI_Iprobe
MPI_Irecv
MPI_Irsend
MPI_Isend
MPI_Issend
MPI_Keyval_create
MPI_Keyval_free
MPI_Pack
MPI_Probe

MPI_Recv
MPI_Recv_init
MPI_Reduce
MPI_Reduce_scatter
MPI_Request_free
MPI_Rsend
MPI_Rsend_init
MPI_Scan
MPI_Scatter
MPI_Scatterv
MPI_Send
MPI_Send_init
MPI_Sendrecv
MPI_Sendrecv_replace
MPI_Ssend
MPI_Ssend_init
MPI_Start
MPI_Startall
MPI_Test
MPI_Testall
MPI_Testany
MPI_Testsome
MPI_Topo_test
MPI_Type_commit
MPI_Type_free
MPI_Type_get_contents
MPI_Type_get_envelope
MPI_Unpack
MPI_Wait
MPI_Waitall
MPI_Waitany
MPI_Waitsome
MPI_Win_complete
MPI_Win_create
MPI_Win_fence
MPI_Win_free
MPI_Win_get_group
MPI_Win_lock
MPI_Win_post
MPI_Win_start
MPI_Win_test
MPI_Win_unlock
MPI_Win_wait

Top

MPI Routines For Which mpiP Gathers Sent Message Size Data

MPI_Allgather
MPI_Allgatherv
MPI_Allreduce
MPI_Alltoall
MPI_Bcast
MPI_Bsend
MPI_Gather
MPI_Gatherv
MPI_Ibsend
MPI_Irsend
MPI_Isend
MPI_Issend
MPI_Reduce
MPI_Rsend
MPI_Scan
MPI_Scatter
MPI_Send
MPI_Sendrecv
MPI_Sendrecv_replace
MPI_Ssend

Top

MPI Routines For Which mpiP Gathers I/O Data

MPI_File_close

MPI_File_open
MPI_File_preallocate
MPI_File_read
MPI_File_read_all
MPI_File_read_at
MPI_File_seek
MPI_File_set_view
MPI_File_write
MPI_File_write_all
MPI_File_write_at

Top

MPI Routines For Which mpiP Gathers RMA Origin Data

MPI_Accumulate
MPI_Get
MPI_Put

Top

How to add MPI calls to profile

Here is an example of how to add MPI calls to mpiP, using the MPI_Comm_spawn call as an example:

1. Insert the appropriate call with appropriate arguments into the mpi.protos.txt.in file:

   ```
   int MPI_Comm_spawn ( char *command, char *argv[], int maxprocs, MPI_Info info, int root,
   MPI_Comm comm, MPI_Comm *intercomm, int array_of_errcodes[] )
   ```

2. Configure mpiP or, if you have already configured mpiP, run `./config.status`.
3. Currently, it is necessary to add entries for MPI opaque objects to the make-wrappers.py script. MPI_Comm_spawn has 3 arguments that are MPI opaque object which need to be added to make-wrappers.py dictionaries:

   1. Add the following entries to the opaqueInArgDict:

      ```
      ("MPI_Comm_spawn", "info"):"MPI_Info",
      ("MPI_Comm_spawn", "comm"):"MPI_Comm",
      ```

   2. Add the following entry to the opaqueOutArgDict:

      ```
      ("MPI_Comm_spawn","intercomm"):"MPI_Comm",
      ```

4. When you build mpiP, you should see an MPI_Comm_spawn wrapper in the generated wrappers.c file.

Top

License

Top

For further information please send mail to mpip-help@lists.sourceforge.net.

Last modified on March 13th, 2014.
UCRL-CODE-223450