mpiP 3.5

A light-weight MPI profiler.

Introduction

mpiP is a light-weight 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.

Downloading

The current version of mpiP can be accessed at https://github.com/LLNL/mpiP/releases/latest.

New Features & Bug Fixes

Version 3.5 includes several new features, including

- Multi-threaded support
- Additional MPI-IO functions
- Various updates including
    - New configuration options and tests
    - Updated test suite
    - Updated build behavior

Please see the ChangeLog for additional changes.

Configuring and Building mpiP

Dependencies

- MPI installation
- libunwind : for collecting stack traces.
- binutils : for address to source translation
- glibc backtrace() can also be usef for stack tracing, but source line numbers may be inconsistent.

Configuration

Several specific configuration flags can be using, as provided by `./configure -h`. Standard configure flags, such as CC, can be used for specifying MPI compiler wrapper scripts.

Build Make Targets

Target	Effect
[default]	Build libmpiP.so
all	Build shared library and all tests
check	Use dejagnu to run and evaluate tests

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.

Instrumentation

Link Time Instrumentation

Link the mpiP library with an executable. The dependent libraries may need to be specified as well. If the link command includes the MPI library, order the mpiP library before the MPI library, as in `-lmpiP -lmpi`.

Run Time Instrumentation

An uninstrumented executable may able to be instrumented at run time by setting the

LD_PRELOAD environment variable, as in `export LD_PRELOAD=[path to mpiP]/libmpiP.so`. Preloading libmpiP can possibly interfere with the launcher and may need to be specified on the launch command, such as `srun -n 2 --export=LD_PRELOAD=[path to mpiP]/libmpiP.so [executable]`.

mpiP Run Time Flags

The behavior of mpiP can be set at run time through the use of the following flags. Multiple flags can be delimited with spaces or commas.

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 .	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 the MPIP variable.

mpiP Output

Header information provides basic information about your performance experiment.

```
@ mpiP
@ Command : /g/g0/chcham/mpiP/Testing/tests/AMG/./test/amg -P 4 2 2 -n 50
 50 50
@ Version                  : 3.5.0
@ MPIP Build date          : Oct 20 2020, 18:22:06
@ Start time               : 2020 10 20 18:25:41
@ Stop time                : 2020 10 20 18:25:45
@ Timer Used               : PMPI_Wtime
@ MPIP env var             : -k3,-y
@ Collector Rank           : 0
@ Collector PID            : 9164
@ Final Output Dir         : .
@ Report generation        : Single collector task
@ MPI Task Assignment      : 0 surface101
@ MPI Task Assignment      : 1 surface101
@ MPI Task Assignment      : 2 surface101
@ MPI Task Assignment      : 3 surface101
```

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       9.51      0.168     1.76
   1       9.51      0.168     1.76
   2       9.51      0.228     2.40
   3       9.51      0.219     2.31
   *       38.1      0.783     2.06
```

The callsite section identifies all the MPI callsites within the application. The first number is the callsite ID for this mpiP file, followed by the stack trace level. The line number, parent function, and MPI function. Note that the default setting for callsite stack walk depth is 1. The MPIP run time flag -k can control the number of stack frames per callsite that are provided in the report.

```
---------------------------------------------------------------------
--
@--- Callsites: 211 ----------------------------------------------------
--
---------------------------------------------------------------------
--
 ID Lev File/Address             Line Parent_Funct
  1   0 mpistubs.c                1172 hypre_MPI_Allreduce
  1   1 timing.c                   338 hypre_PrintTiming
  1   2 amg.c                      421 main
  2   0 mpistubs.c                1128 hypre_MPI_Testall
  2   1 exchange_data.c            413 hypre_DataExchangeList
  2   2 new_commpkg.c              272 hypre_NewCommPkgCreate_core
```

The aggregate time section is a 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%      Count    COV
Isend                  25        926    1.45   16.06      71742   0.20
Irecv                  55        915    1.43   15.86      71742   0.19
Waitall               186        648    1.01   11.24       7722   0.55
Allreduce             174        346    0.54    6.00        336   0.51
Isend                 112        173    0.27    2.99      13332   0.22
Irecv                 178        170    0.27    2.95      13332   0.21
Irecv                  71        137    0.22    2.38      10802   0.21
```

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

```
---------------------------------------------------------------------
--
@--- Aggregate Sent Message Size (top twenty, descending, bytes) --------
--
---------------------------------------------------------------------
--
Call                 Site      Count      Total       Avrg  Sent%
Isend                  25      71742   1.47e+08   2.04e+03  63.34
Isend                 112      13332   2.99e+07   2.24e+03  12.91
Isend                 155       1068   1.16e+07   1.09e+04   5.02
Isend                  84       1530   6.03e+06   3.94e+03   2.60
Isend                  47       4126   4.69e+06   1.14e+03   2.03
```

If collective histograms are enabled (MPIP=-y), the following section provides histogram data for each collective MPI call, reporting the percent of the total MPI collective time for specific comm size and data size bins.

```
---------------------------------------------------------------------
--
@--- Aggregate Collective Time (top twenty, descending) ----------------
--
---------------------------------------------------------------------
--
Call                 MPI Time %             Comm Size             Data Si
ze
Allreduce                 0.182         16 -       31          8 -
15
Allreduce                0.0566         16 -       31          0 -
 7
Bcast                    0.0155         16 -       31          0 -
 7
Bcast                   0.00444         16 -       31          8 -
15
```

If point-to-point histograms are enabled (MPIP=-p), the following section provides histogram data for each sending MPI call, reporting the percent of the total MPI point-to-point data sent for specific comm size and data size bins.

```
---------------------------------------------------------------------
--
@--- Aggregate Point-To-Point Sent (top twenty, descending) -------------
--
---------------------------------------------------------------------
--
Call                 MPI Sent %             Comm Size             Data Si
ze
Isend                      69.5         16 -       31      16384 -    327
67
Isend                      10.7         16 -       31       8192 -    163
83
Isend                      7.21         16 -       31       1024 -     20
47
Isend                      3.84         16 -       31        256 -      5
11
Isend                      2.99         16 -       31        512 -     10
23
Isend                      1.96         16 -       31      32768 -    655
35
```

If the final sections have not been suppressed (MPIP=-d), they report 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): 807 --------------
--
---------------------------------------------------------------------
--
Name              Site Rank  Count      Max     Mean      Min   App%   MP
I%
Allreduce            1    0      1   0.0138   0.0138   0.0138   0.00   0.
01
Allreduce            1    1      1   0.0138   0.0138   0.0138   0.00   0.
01
Allreduce            1    2      1   0.0143   0.0143   0.0143   0.00   0.
01
Allreduce            1    3      1    0.013    0.013    0.013   0.00   0.
01
Allreduce            1    *      4   0.0143   0.0137    0.013   0.00   0.
01
```

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       S
um
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
```

where

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       S
um
File_read            1    0      20        64        64        64      12
80
File_read            1    1      20        64        64        64      12
80
File_read            1    *      40        64        64        64      25
60
```

Controlling Profiling Scope

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 ... */
```

MPI Routines Profiled with mpiP

```
MPI_Accumulate
MPI_Allgather
MPI_Allgatherv
MPI_Allreduce
MPI_Alltoall
MPI_Alltoallv
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_compare
MPI_Comm_create
```

```
MPI_Comm_create_errhandler
MPI_Comm_create_keyval
MPI_Comm_delete_attr
MPI_Comm_dup
MPI_Comm_free
MPI_Comm_free_keyval
MPI_Comm_get_attr
MPI_Comm_get_errhandler
MPI_Comm_group
MPI_Comm_rank
MPI_Comm_remote_group
MPI_Comm_remote_size
MPI_Comm_set_attr
MPI_Comm_set_errhandler
MPI_Comm_size
MPI_Comm_split
MPI_Comm_test_inter
MPI_Compare_and_swap
MPI_Dims_create
MPI_Errhandler_free
MPI_Error_class
MPI_Error_string
MPI_Fetch_and_op
MPI_File_close
MPI_File_delete
MPI_File_get_amode
MPI_File_get_byte_offset
MPI_File_get_group
MPI_File_get_info
MPI_File_get_position
MPI_File_get_size
MPI_File_get_view
MPI_File_open
MPI_File_preallocate
MPI_File_read
MPI_File_read_all
MPI_File_read_at
MPI_File_read_at_all
MPI_File_seek
MPI_File_set_info
MPI_File_set_size
MPI_File_set_view
MPI_File_sync
MPI_File_write
MPI_File_write_all
```

```
MPI_File_write_at
MPI_File_write_at_all
MPI_Finalize
MPI_Finalized
MPI_Gather
MPI_Gatherv
MPI_Get
MPI_Get_accumulate
MPI_Get_address
MPI_Get_count
MPI_Get_elements
MPI_Get_processor_name
MPI_Get_version
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_range_excl
MPI_Group_range_incl
MPI_Group_rank
MPI_Group_size
MPI_Group_translate_ranks
MPI_Group_union
MPI_Iallgather
MPI_Iallgatherv
MPI_Iallreduce
MPI_Ialltoall
MPI_Ialltoallv
MPI_Ialltoallw
MPI_Ibarrier
MPI_Ibcast
MPI_Ibsend
MPI_Iexscan
MPI_Igather
MPI_Igatherv
MPI_Init
MPI_Init_thread
```

```
MPI_Initialized
MPI_Intercomm_create
MPI_Intercomm_merge
MPI_Iprobe
MPI_Irecv
MPI_Ireduce
MPI_Ireduce_scatter
MPI_Ireduce_scatter_block
MPI_Irsend
MPI_Iscan
MPI_Iscatter
MPI_Iscatterv
MPI_Isend
MPI_Issend
MPI_Op_create
MPI_Op_free
MPI_Pack
MPI_Pack_size
MPI_Probe
MPI_Put
MPI_Raccumulate
MPI_Recv
MPI_Recv_init
MPI_Reduce
MPI_Reduce_scatter
MPI_Request_free
MPI_Rget
MPI_Rget_accumulate
MPI_Rput
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_Test_cancelled
MPI_Testall
```

```
MPI_Testany
MPI_Testsome
MPI_Topo_test
MPI_Type_commit
MPI_Type_contiguous
MPI_Type_count
MPI_Type_create_darray
MPI_Type_create_hindexed
MPI_Type_create_hvector
MPI_Type_create_indexed_block
MPI_Type_create_struct
MPI_Type_create_subarray
MPI_Type_free
MPI_Type_get_contents
MPI_Type_get_envelope
MPI_Type_get_extent
MPI_Type_indexed
MPI_Type_size
MPI_Type_vector
MPI_Unpack
MPI_Wait
MPI_Waitall
MPI_Waitany
MPI_Waitsome
MPI_Win_allocate
MPI_Win_allocate_shared
MPI_Win_attach
MPI_Win_complete
MPI_Win_create
MPI_Win_create_dynamic
MPI_Win_detach
MPI_Win_fence
MPI_Win_flush
MPI_Win_flush_all
MPI_Win_flush_local
MPI_Win_flush_local_all
MPI_Win_free
MPI_Win_get_group
MPI_Win_get_info
MPI_Win_lock
MPI_Win_lock_all
MPI_Win_post
MPI_Win_set_info
MPI_Win_shared_query
MPI_Win_start
```

```
MPI_Win_sync
MPI_Win_test
MPI_Win_unlock
MPI_Win_unlock_all
MPI_Win_wait
MPI_Wtick
MPI_Wtime
```

Contributors

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

- Jeffrey Vetter (Oak Ridge National Laboratory)
- Michael McCracken (UCSD)
- Chris Chambreau (Lawrence Livermore National Laboratory)
- 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)
- Jeff Hammond (Intel)
- Artem Polyakov (Mellanox)
- Greg Lee (Lawrence Livermore National Laboratory)
- Rob Latham
- Josh Milthorpe (Australian National University Research School of Computer Science)

License

- Redistributions of source code must retain the above copyright notice, this list of conditions and the disclaimer below.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the disclaimer (as noted below) in the documentation and/or other materials provided with the distribution.

- Neither the name of the UC/LLNL nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OF THE UNIVERSITY OF CALIFORNIA, THE U.S. DEPARTMENT OF ENERGY OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Additional BSD Notice

1. This notice is required to be provided under our contract with the U.S. Department of Energy (DOE). This work was produced at the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-ENG-48 with the DOE.

2. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately-owned rights.

3. Also, reference herein to any specific commercial products, process, or services by trade name, trademark, manufacturer or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.