MPInside Reference Guide

007–5780–001

Record of Revision

Version	Description
001	June 2011 Original Printing.

Contents

About This Manual . . . . **xi**
Obtaining Publications xi
Related Publications and Other Sources xi
Conventions xi
Reader Comments xii

1. MPInside Profiling Tool **1**
Overview 1
Non-synchronized Send/Receive Pair Definition and Terminology 2
Using MPInside Tool 5
SGI MPT 5
X86 Intel MPI 6
X86 HP-MPI 6
X86 SCALI MPI 6
Post-Processing 6
Restraining the Profile to Selected Parts of the Application 7
Environment Variables and Stack of Features 8
`MPINSIDE_EVAL_COLLECTIVE_WAIT` 8
`MPINSIDE_EVAL_SLT` 9
`MPINSIDE_WAIT_TIME_NO_CROSSREF` 9
`MPINSIDE_CALLSTACK_DEPTH <integer number>` 9
`MPINSIDE_CROSS_REFERENCE` 9
`MPINSIDE_LITE` 9
`MPINSIDE_MATRICES [EXA | PLA | P2P:[+|-B][S|M]` 10
`MPINSIDE_SIZE_DISTRI [T+]nb_bars[:first-last]` 10

MPINSIDE_WITH_PERFSUITE : ALL|OUT (X86-64 only) 15
MPINSIDE_PCL_EVENTS : [O|A@]<PCL events list> 16
Miscellaneous Environment Variables 16
MPINSIDE_CALLSTACK_MAX_RECV_ENTRIES <Integer value> 16
MPINSIDE_CALLSTACK_MAX_SEND_ENTRIES <Integer value> 17
MPINSIDE_CALLSTACK_MAX_WAIT_ENTRIES <Integer value> 17
MPINSIDE_CALLSTACK_SKIP 17
MPINSIDE_COLLECTIVE_WINDWOW <MPI_collective_name>:<START>:<END>: . 17
MPINSIDE_COMM_TO_WATCH <Integer value> 17
MPINSIDE_COMM_T_W <Integer value> 18
MPINSIDE_NON_STOPPING_WINDOW 18
MPINSIDE_SHOW_W <Integer value> 18
MPINSIDE_CUT_OFF <real value> 18
MPINSIDE_DELAY_AT_INIT <Integer value> 18
MPINSIDE_ING_COLLECTIVE_BRANCHES 19
MPINSIDE_INTERNAL_TAG_START <Integer value> 19
MPINSIDE_LIB: <MPT|IMPI|HPMPI|SCALIMPI> 19
MPINSIDE_BINARY_MATRICES_DIR Directory 19
MPINSIDE_MAT_START_STOP <start float value: start stop value> . . 19
MPINSIDE_OUTPUT_PREFIX 19
MPINSIDE_PARTIAL_EXPERIMENT 20
MPINSIDE_PRINT_ALL_COLUMNS 20
MPINSIDE_PRINT_DIRTY 20
MPINSIDE_PRINT_SIZ_IN_K 20
MPINSIDE_SHOW_READ_WRITE 21
MPINSIDE_TRANSLATE_PERSISTENTS <Nb_entries, default 128> 21
mpinside_stats File 21
User Counters 22

Columns Meaning 23
Bytes Transferred and Number of Requests Arrays 24
Point to Point Function 24
Collective functions 25
`mpinside_stats_.M-N` files 26
MPInside Binary Transfer Matrices 29
`pram` Utility 30
`mpinside_clstk_post.xxx` Files 30
Partner Branch Kinds 35
Ordinary Branches 35
Send Branches 35
Recv branches 36
Wait Branches 36
Communication "Stiffness" 38
Perfect Interconnect Zero Latency Infinite Bandwidth 40

2. Using the MPInside Profiling Tool 45
Selecting a Window of Observation using Re-compilation 45
Selecting a Window of Observation with a Collective Function Heartbeat 46
Spy the Collective Functions 46

Appendix A. MPInside(3) Man Page 49

Index 65

Figures

Figure 1-1 Non-synchronized Send/Receive Pair Communication 3
Figure 1-2 Histogram of the Request Sizes Distribution 12
Figure 1-3 Crash Application with 64 CPUs Running SGI MPI 1 of 2 13
Figure 1-4 Crash Application with 64 CPUs Running SGI MPI 2 of 2 14
Figure 1-5 LINPACK Run on an SGI Altix ICE System 15
Figure 1-6 Array of Values Printed in the `mpinside_stats` File 22
Figure 1-7 `mpinside_stats` User Counters 22
Figure 1-8 PoP2 Wait Time Matrix 28
Figure 1-9 PoP2 Time Transfer Only 29
Figure 1-10 Pop2 (64 CPU run) CPU 9 Dominant Recv Branch 34
Figure 1-11 Communication Stiffness Chart 1 of 2 38
Figure 1-12 Communication Stiffness Chart 2 of 2 39
Figure 1-13 PARATEC Application on Altix ICE 1 of 2 41
Figure 1-14 PARATEC Application on Altix ICE 2 of 2 42
Figure 1-15 LINPACK Measure versus Perfect Interconnect Timing 43

About This Manual

This publication documents the SGI MPInside MPI profiling tool.

Obtaining Publications

You can obtain SGI documentation in the following ways:

- See the SGI Technical Publications Library at: http://docs.sgi.com. Various formats are available. This library contains the most recent and most comprehensive set of online books, release notes, man pages, and other information.
- You can also view man pages by typing `man` *title* on a command line.

Related Publications and Other Sources

This section describes documentation you may find useful, as follows:

- *Message Passing Toolkit (MPT) User's Guide*

  Describes industry-standard message passing protocol optimized for SGI computers.

- *SGI Performance Suite 1.x Start Here*

  Provides information about the SGI Performance Suite 1.x release. Provides descriptions of current SGI software and hardware manuals.

Conventions

The following conventions are used throughout this document:

Convention	**Meaning**
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.

`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names.
variable	Italic typeface denotes variable entries and words or concepts being defined.
`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. (Output is shown in nonbold, fixed-space font.)
[ ]	Brackets enclose optional portions of a command or directive line.
...	Ellipses indicate that a preceding element can be repeated.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this publication, contact SGI. Be sure to include the title and document number of the publication with your comments. (Online, the document number is located in the front matter of the publication. In printed publications, the document number is located at the bottom of each page.)

You can contact SGI in any of the following ways:

- Send e-mail to the following address:

  techpubs@sgi.com

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

- Send mail to the following address:

  SGI
  Technical Publications
  46600 Landing Parkway
  Fremont, CA 94538

SGI values your comments and will respond to them promptly.

MPInside Profiling Tool

This chapter describes MPInside, which is an MPI profiling tool developed by SGI.

Overview

Note: The prefix name of the statistic files resulting from the MPInside command can be chosen by the user. This document uses the default prefix `mpinside`.

MPInside is an SGI MPI profiling tool that provides valuable information for MPI application developers to optimize their application. It helps developers figure out where the MPI Send / Receive pairs are not executed synchronously. With non-synchronized Send / Receive pairs, the MPI communications can be very slow, independent of the power of the underlying MPI library / hardware engine. For many MPI applications, the MPI communication times are due to the lack of synchronizations of these Send / Receive pairs rather than the speed of the underlying MPI / hardware engine. MPInside measures this un-synchronized time for all the MPI ranks involved in the application for all the MPI functions activated. It also allows you to tell the actual speed at which the MPI engine did such communications, measured as the ratio Bytes received / (time of the MPI function minus the synchronization time) accumulated per CPU, as well as, in a CPU x CPU matrix. In addition, MPInside precisely reports the timing described above on a branch basis, automatically. A branch is an MPI function, with all of its ancestors in the calling sequence. MPInside provides the routine name and the source file line number for all the routines defining a branch.

All branches are put in relation with the other CPU branches that had a Send / Receive partnership with them. For any CPU, any Received branch performed by that CPU has partners. A partner set is described by four numbers:

- Sending rank number
- Sending CPU branch identification
- Percentage of time accounted to this partnership, in regard to the total execution wait time of this Received branch
- Percentage of the execution wait time attributed to the lack of synchronization.

The aim of MPInside is to tell you where and how much non-synchronized communication occurred in your application degrading application performance. In addition to simple measurement, MPInside is able to model the communications. Knowing how MPI communication latency and bandwidth changes affect your application can help you improve its performance.

The `MPInside`(3) man page contains detailed information on using the MPInside profiling tool. To see the `MPInside`(3) man page on-line, make sure the `MPInside/3.3` module is available and loaded, as follows:

```
uv44-sys:~ # module avail

------------------------ /usr/share/modules/modulefiles ------------------------
MPInside/3.3       module-info        null               sgi-upc/1.05
chkfeature         modules            perfboost          sgi-upc-devel/1.05
dot                mpiplace/1.01      perfcatcher        use.own
module-cvs         mpt/2.04           scotch/5.1.11
uv44-sys:~ # module load MPInside/3.3
```

To see a copy of the `MPInside`(3) man page, perform the following:

```
uv44-sys:~ # man MPInside
```

For your convenience, you can find a copy of the `MPInside`(3) man page at Appendix A, "MPInside(3) Man Page" on page 49.

Non-synchronized Send/Receive Pair Definition and Terminology

Figure 1-1 on page 3 shows an example of non-synchronized communication between a Send / Receive pair.

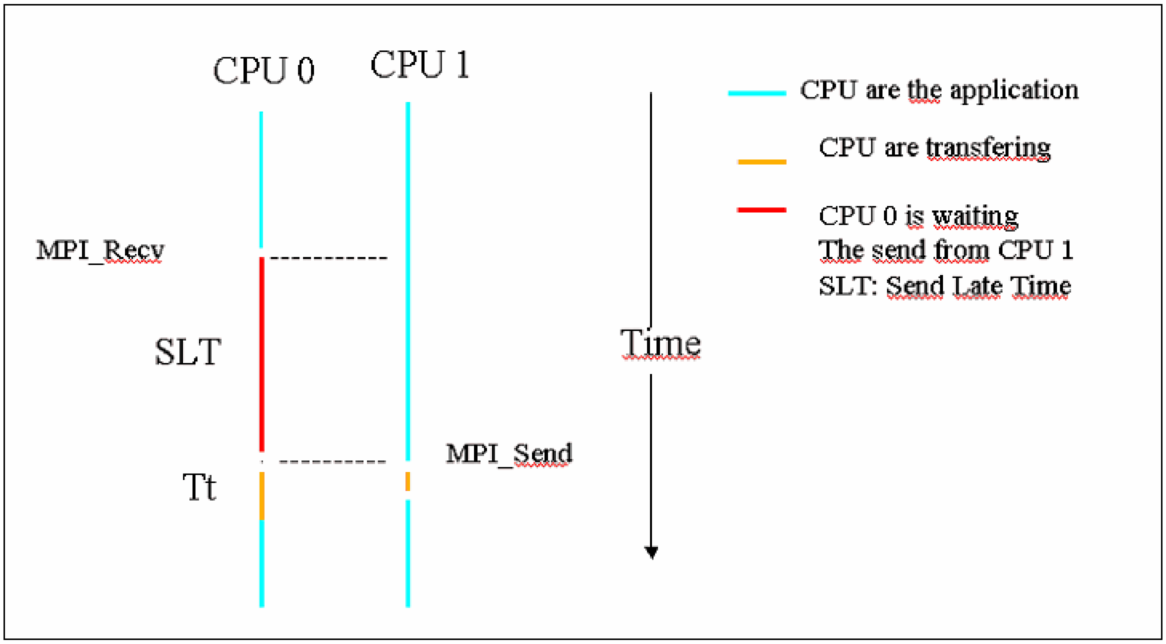


Figure 1-1 Non-synchronized Send/Receive Pair Communication

This section describes MPI communication terminology, as follows:

- *Function time* (FT)

  The time before the call to the MPI function minus time when returning to the application. This time is equal to SLT + Tt in Figure 1-1 on page 3.

- The *Transfer time* (Tt)

  The time when the data is actually being transferred (see Figure 1-1 on page 3).

- The *Function Waiting time* (FWT)

  In Figure 1-1 on page 3, this time is equal to the FT time because `MPI_Recv` is a blocking function. For a non-blocking function, such as `MPI_Irecv`, FWT is the time of the `MPI_Wait` function that "finished" the request (in the MPI sense) corresponding to this function.

- The *Send late time* (SLT) is computed as the difference between the time when the corresponding send entered the MPI send function and the time when the Wait for the receive was performed.
- A *branch* is defined by a sequence of calls terminated by an MPI function. A branch has a unique identification number. Such a number could differ from one CPU to the other even if both refer to exactly the same sequence of calls. The identification depends on the order they are encountered in the MPInside library. Some branches have *partners*.
- A `partner` is defined by four numbers : A, #B, C, D. There numbers are defined, as follows:
  - A

    Rank number that did the MPI `Send`/`Isend` for this branch.
  - #B

    MPI_Send/Isend Branch ident
  - C

    Percent of this `MPI_Recv` that involved this "A" rank "#B" MPI Send branch.
  - D

    Percent of this `MPI_Recv` where the corresponding Send was arriving late.
  - *Ordinary branches*

    Ordinary branches do not have partners nor are they targets of another branch. Collective function branches are of this type.
- *Send branches*

  Send branches do not get partners but are targets of "Receive branches" or "Wait branches".
- *Wait branches*

  Wait branches connect, as partners, the "Send/Recv branches" that initiated the MPI requests.

  Each "Wait branch" precisely reports, for all the "Send/Recv branches" that were connected to it, the percentage of (FWT) time to account to a particular

"Send / Recv branch" in regard to the total execution time of this "Wait branch". An `MPI_Wait` branch is a Wait branch as well as a `MPI_Recv` branch, for example.

- *Recv branches*

  Have Send partners and are target of "Wait branches".

  Each "`Recv` branch" precisely reports, for all the "Send branches" that were with it, as follows:

  - The ranks of such Sends
  - The Send branch IDs
  - The percentage of execution time (FWT) to account to this particular "Send branch" in regard to the total WAIT time of this "`Recv` branch".
  - The percentage of time (SLT) such "Send Branches" were arriving late in regard to the matching Receive posting.

For more detailed information about branches, see the `mpinside_clsth_post.xxx` files.

Using MPInside Tool

The MPInside tool does not require any changes to your application, however, it provides more information if the application was compiled with the `-g` flag. You need to set the appropriate environment variable depending on which flavor of MPI you are running.

SGI MPT

For SGI MPT, perform the following:

```
setenv MPINSIDE_LIB MPT
# this is the default you for MPT so you do not need to set
this variable mmpirun -np xxx MPInside your_prog [your_args]
```

X86 Intel MPI

For X86 Intel MPI, perform the following:

```
setenv MPINSIDE_LIB IMPI
mpirun -np xx MPInside your_prog [your_args]
```

X86 HP-MPI

For HP-MPI, perform the following:

```
setenv MPINSIDE_LIB HPMPI
mpirun -np xx MPInside your_prog [your_args]
```

X86 SCALI MPI

For X86 SCALI MPI, perform the following:

```
setenv MPINSIDE_LIB SCALIMPI
# you need to specify the full path for MPInside
mpirun -np xx INSTALL_DIR/bin/MPInside your_prog [your_args]
```

Note: HP-MPI was acquired by Platform Computing, Inc. It has been combined with Scali-MPI, as described on the Platform Computing web site: “Platform MPI 8.1 combines the broad adoption and scalability of HP-MPI with the performance of Scali-MPI and is fully compliant with the MPI 1.2 and 2.2 standards.”

Platform MPI uses the same `mpi.h` include file as the product fomerly called Scali-MPI. You can use MPInside with Platform MPI. Use the environment variable `setenv MPINSIDE_LIB SCALIMPI`, as described above.

Post-Processing

At the end of the run, you will get at least one `mpinside_stats` file and if the correct feature is activated, a set of `mpinside_stats.N_M` files. When running with less that 255 CPUs, only one `mpinside_stats.0-254` file is produced. These files are described below. When the `MPINSIDE_CALLSTACK_DEPTH` variable is set to a value greater than zero, you will get one file per rank, `xx`, named `mpinside_clstk.xx` . These `mpinside_clstk.xx` files need to be post processed by the `MPInside_post` command which builds a set of `mpinside_clstk_post.xx` files described below. The `MPInside_post` syntax is, as follows:

```
MPInside_post [-s starting_rank] [-e ending_rank]  [-c cutoff] [-l] \
                  [-a]report_prefix
          -s starting_rank: default = 0
          -e ending_rank  : default = 0
          -l              : print source file line numbers if available
                          : in the binaries. The default is to only
                            print routine names
          -a              : print Recv Branch partners as a set of
A:#B:C:D.(default)
                           A : CPU number
                           B : Send branch Id
                           C : Wait time (FWT) due to this send branch
                              (%total wait branch)
                           D : Time (SLT) this send was late (% of C)
                           If -a is set print an array of a:#B lines
                           with C and D columns with values in second instead of %
          -c cutoff       : discard line that are responsible of
                            cutoff % of the total MPI time default = 0.50 %
                            report ouputs are named report_prefix_post.xx
```

Restraining the Profile to Selected Parts of the Application

MPInside provides ways to restrain the profile to selected parts of the application. This can be done without re-compilation using the functionality activated by the environment variable `MPINSIDE_COLLECTIVE_WINDOW` or by inserting calls inside the application source code (`mpinside_start()` and `mpinside_end()`), along with the setting of the environment variable `MPINSIDE_PARTIAL_EXPERIMENT`. By default, the application is terminated when the window set by these mechanism is complete. Inside the window of observation, (which can be the whole program, see Chapter 2, "Using the MPInside Profiling Tool" on page 45), collecting statistics can be suspended by calling `mpinside_suspend()` and then resumed by calling `mpinside_resume()`. These four calls MUST be collective calls involving all ranks. You must ensure that after calling the `mpinside_start()`, `mpinside_end()`, `mpinside_suspend()` and `mpinside_resume()` that no pending MPI requests still remain. For example, a request generated by a `MPI_Irecv` call before calling one of the four functions without the corresponding `MPI_Wait` being called. Unpredictable results may happen if this constraint is not respected. When building the binary, link with libMPInside_stub.so. This library must be in a directory listed in the LD_LIBRARY_PATH variable if the built binary is not run prefixed by the MPInside launcher. In this case, the four functions described above will have no effect. It is a fatal error to call mpinside_resume() if mpinside_suspend() was not previously called. It is a fatal error to call mpinside_suspend() if the profiling was not started. For example, if MPINSIDE_PARTAIL_EXPERIMENT is set, it is a fatal error to call mpinside_suspend() before calling mpinside_start().

Environment Variables and Stack of Features

Most of the MPInside features can be stacked. This section describes the environment variables that command that stack, ordered by the least amount of information to the highest amount of information.

By default (with no environment variables set), MPInside creates at least a file named mpinside_stats. This file contains five set of columns which can be easily exploited by a spreadsheet like Excel, as follows:

- Set 1: Time outside MPI + all the MPI functions timing
- Set 2-3: Amount of char transmitted plus the number of requests with the Send attribute
- Set 4-5: Same as Set 2-3 but with the Recv attribute

For more information on the mipinside_stats file, see "mpinside_stats File" on page 21.

MPINSIDE_EVAL_COLLECTIVE_WAIT

If set, MPInside puts an MPI_Barrier (and times it) before any MPI collective operation. This assumes that the time of a collective operation is the time of all processors to synchronize plus the time of the operation. This is not always true, but it is true in most of the cases. The time to really perform the collective operation is very short compared to the synchronization time. In the mpnside_stats file, the column "b_xxx" will give the MPI_barrier time of the corresponding "xxx" MPI collective function and "xxx" the remaining time. When MPINSIDE_PARTNER_MATCH is set to TOKEN, setting MPINSIDE_EVAL_COLLECTIVE_WAIT will also lead to evaluate the "Stiffness" of the application (see "Communication "Stiffness"" on page 38).

MPINSIDE_EVAL_SLT

If set, MPInside will measure the time the Sends were late (SLT) compared to the Recv-Wait arrivals. Such time will be w_xxx in the mpnside_stats file. xxx could, for example, be MPI_Wait or MPI_Recv. It cannot be MPI_Irecv, because the Send late time, if any, will be for this last function accounted in an MPI_Wait-like function. MPINSIDE_EVAL_SLT is equivalent to MPINSIDE_CALLSTACK_DETH equals one plus MPINSIDE_CROSS_REFERENCE, except no mpinside_clstk.xxx files are created.

MPINSIDE_WAIT_TIME_NO_CROSSREF

MPINSIDE_WAIT_TIME_NO_CROSSREF operation is deprecated, use MPINSIDE_EVAL_SLT instead.

MPINSIDE_CALLSTACK_DEPTH <integer number>

If set, MPInside will unwind the stack up to the depth specified and a set of mpinside_clstk.xxx files will be created (one per rank). These files will contain statistics about all the branches (see definition above) that have an MPI function as leaf. The mpinside_clstk.xxx files only contain raw addresses. The address-Routine name matching is performed by MPInside_post command that produces mpinside_clstk_post.xxx files (see more information about the format of such files below). If MPInside_post is run with the --l flag, the source file line numbers are also printed (provided the application was compiled with the --g flag). Note that most of the overhead of the tool is imputable to unwind the stack. You should be careful not to set this variable to a number bigger than necessary.

MPINSIDE_CROSS_REFERENCE

If set, MPInside instruments the Branches with "partners" providing timed cross CPU branches information. (See mpinside_clsth_post file.)

MPINSIDE_LITE

The MPInside overhead is very low. Nevertheless, with applications doing a lot of calls to functions like MPI_Test or MPIprobe, the MPInside overhead may be sizeable. With this variable set, the overhead is reduced to a minimum. In this case, only the timings will be reported in the mpinside_stats file. No size and request information will be printed and the only MPInside variables recognized will be MPINSIDE_OUTPUT_PREFIX, MPINSIDE_VERBOSE, MPINSIDE_NON_STOPPING_WINDOW, MPINSIDE_SHOW_READ_WRITE, and MPINSIDE_PARTIAL EXPERIMENT.

MPINSIDE_MATRICES [EXA | PLA | P2P:[+|-B][S|M]

Print transfer matrices files. Default is not to print any matrices files. Option: None: Only point to point operation will be reported. (See mpinside_stats.M_N) below for the format of the output files). The MPINSIDE_MATRICES syntax is, as follows:

EXA	Matrices will include exact P2P transfers implied by Collective functions. (MPT only)
PLA	Matrices will include generic P2P transfers implied by Collective functions. This is the best choice for these matrices to be input to an automatic placement tool.
+B	In addition to the mpinside_stats. M-N. The transfer matrices size and request will be print in binary format to be used as input for the placement tool Sergeant. There will be one file per rank (see the MPINSIDE_BINARY_MATRICES_DIR below and the MPInside binary transfer matrices section).
-B	Binary files will be the only ones produces. Use the pram utility, described in the mpinside_stats. M_N to get ASCII files similar to the mpinside. N-N ones.
S	Collectives and P2P matrices are separated in the binary files.
M	Collectives and P2P matrices are merged in the binary files.

Usage Example

```
setenv MPINSIDE_MATRICES PLA:-B:S
```

MPINSIDE_SIZE_DISTRI [T+]nb_bars[:first-last]

An histogram of the request sizes distribution will be printed at the end of mpinside_stats for rank first to last: Default 12:0-0 (only rank zero and bar size :

0, 128, 256, 512, ... 65536. The cumulus of the calls for all rank is then terminated in the report. This cumulus is always printed even if the variable is not set. If `T+` is specified, each histogram of the request sizes is followed by a size distribution time histogram. On such a histogram, the time taken by functions like `MPI_Wait` or `MPI_Waitall` is not accounted to these functions but to the `MPI_Isend` and `MPI_Irecv` functions that generated the request passed that will next lead to a `Wait` function call.

Usage Example

```
setenv MPINSIDE_SIZE_DISTRI T+12 :0-16
```

Figure 1-2 on page 12 shows an example of the date with the options, above (reduced to CPU 15).

	Sizes	isend	irecv	barrier	bcast	allred	comgroup	comcrea
	65536	0	0	0	0	0	0	0
	32768	0	0	0	0	0	0	0
	16384	0	0	0	0	0	0	0
	8192	0	0	0	0	0	0	0
	4096	0	0	0	0	0	0	0
	2048	72008	72008	0	0	0	0	0
	1024	0	0	0	0	0	0	0
	512	2485096	2485096	0	0	0	0	0
	256	11234010	11234010	0	0	0	0	0
	128	5	5	0	6	0	0	0
	64	0	0	0	59	0	0	0
	32	0	0	0	0	0	0	0
	0	0	0	718	909	821656	3	3
->>	Rank	0	Size	distributior	times<<<			
	Sizes	isend	irecv	barrier	bcast	allred	comgroup	comcrea
	65536	0	0	0	0	0	0	0
	32768	0	0	0	0	0	0	0
	16384	0	0	0	0	0	0	0
	8192	0	0	0	0	0	0	0
	4096	0	0	0	0	0	0	0
	2048	0.201448	1.211336	0	0	0	0	0
	1024	0	0	0	0	0	0	0
	512	1.96275	14.99569	0	0	0	0	0
	256	10.84234	36.72976	0	0	0	0	0
	128	0.000008	0.000008	0	0.000019	0	0	0
	64	0	0	0	0.000209	0	0	0
	32	0	0	0	0	0	0	0
	0	0	0	0.013708	0.007757	19.4962	0.000002	0.000582

Figure 1-2 Histogram of the Request Sizes Distribution

Figure 1-3 on page 13 shows an example of the kind of chart that can be produced from the `mpinside_stats` file using an Excel spreadsheet.

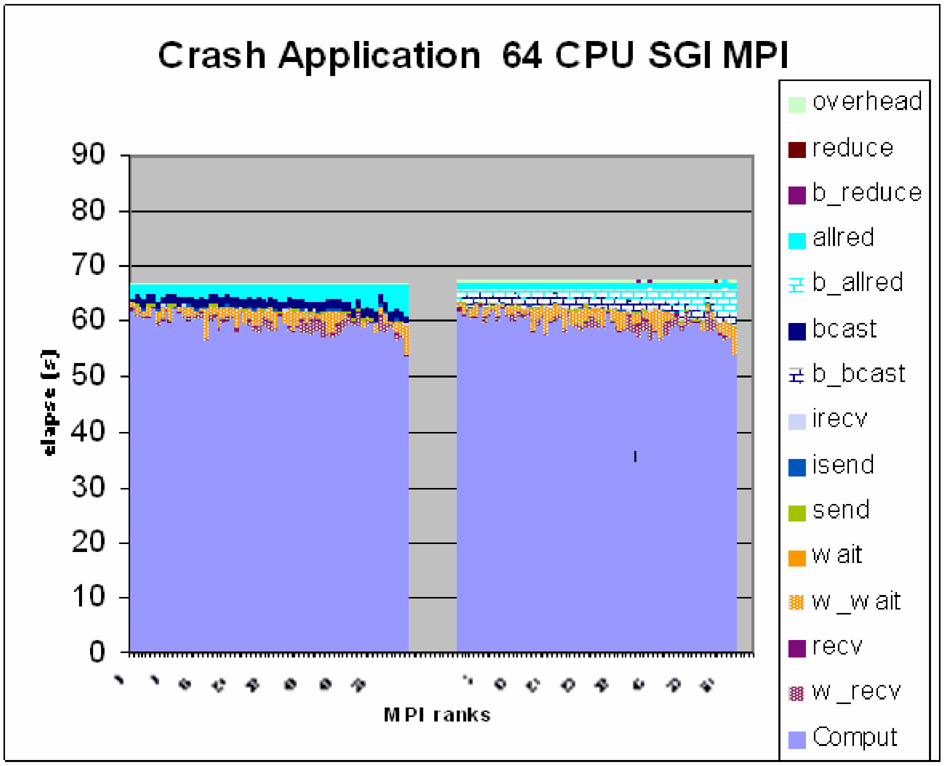


Figure 1-3 Crash Application with 64 CPUs Running SGI MPI 1 of 2

Figure 1-4 on page 14 shows the exact same data as Figure 1-3 on page 13. The only difference is the "Y" axis. The chart on the left on both figures was run with `MPINSIDE_EVAL_SLT`. The charts on the right of both figures were run with the addition of `MPINSIDE_EVAL_COLLECTIVE_WAIT` function.

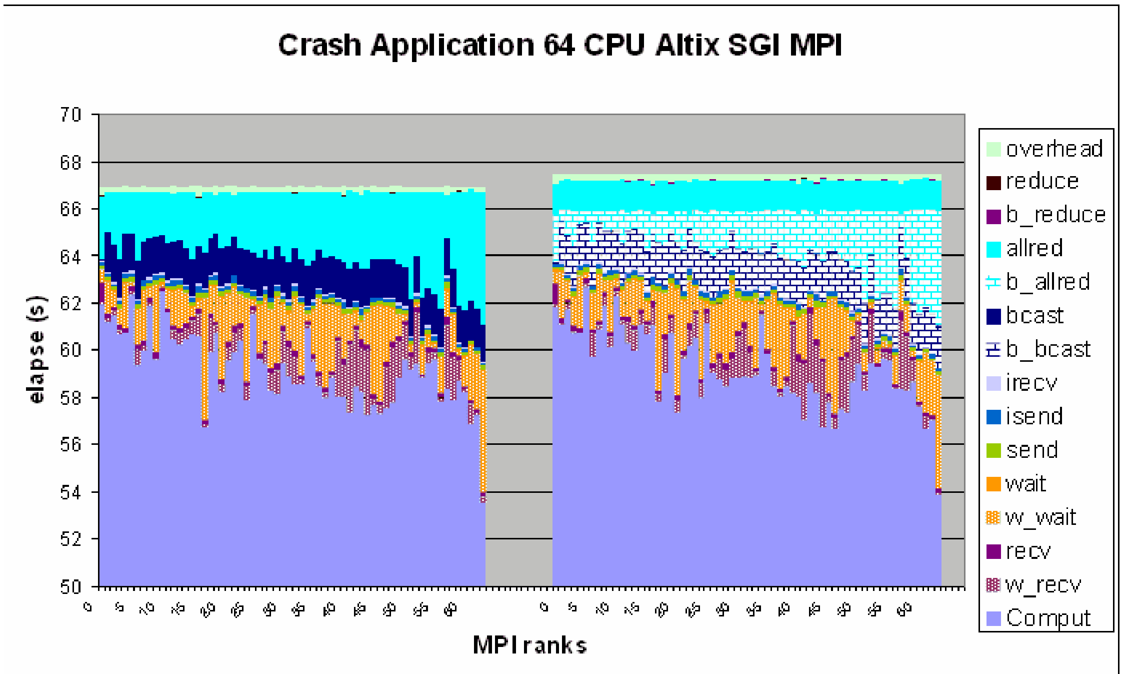


Figure 1-4 Crash Application with 64 CPUs Running SGI MPI 2 of 2

For example, `b_allred` (`b` like barrier or before) in the right charts is the time taken by the `MPI_Barrier` function performed before the `MPI_All_Reduce` (`allred`) function. This `b_allred` time is null in the left charts. `W_Recv` (`W` like `Wait`) is the time the `MPI_Recv` function was blocked just because the associated `Send` was not yet performed by the sending CPU. `Recv` is the time when the data is actually being transferred.

The charts shown in Figure 1-3 on page 13 and Figure 1-4 on page 14 demonstrate the following:

- The MPInside tool induces very little overhead.
- The global MPI profile is not altered because of the MPInside features stacking(Figure 1-3 on page 13).
- Most `MPI_Bcast` and `MPI_Reduce` times are related to process synchronization.

- Most of the `MPI_Recv` and `MPI_wait` times are related to send late arrivals.

Figure 1-5 on page 15 shows another example of a chart that can be made with a few Microsoft Excel operations on columns. This is a LINPACK run on an SGI Altix ICE system with 992 Intel® Xeon® 5320 Series processors (code name Clovertown) running SGI MPT. It shows the bandwidth actually achieved by the MPI/hardware when subtracting the Send late time, so `Tt` as defined, above. This is not at all the bandwidth seen by the application. Rather, this is the true bandwidth the application could expect if all the requests were perfectly synchronized.

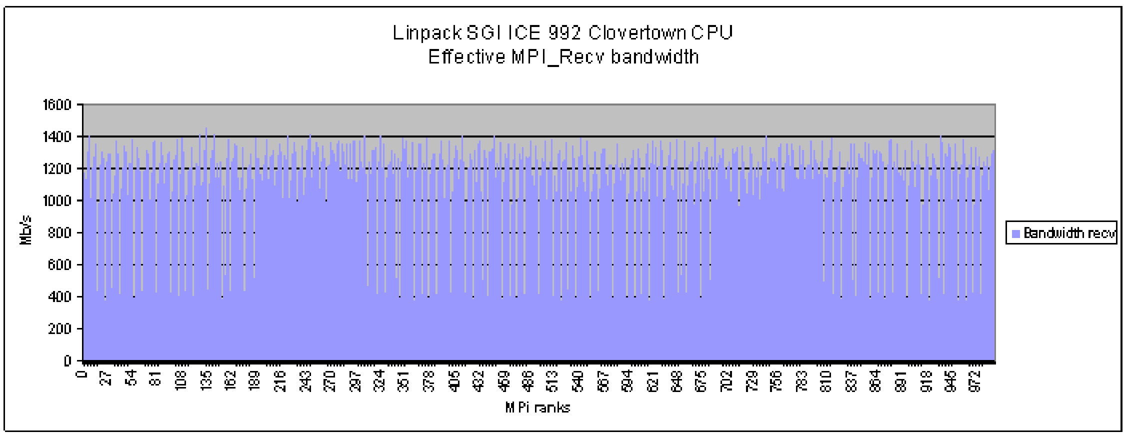


Figure 1-5 LINPACK Run on an SGI Altix ICE System

MPINSIDE_WITH_PERFSUITE : ALL|OUT (X86-64 only)

If set to `ALL`, the Perfsuite profiler will be activated concurrently to the MPInside process for the execution Window. If set to `OUT`, the Perfsuite profiler will be activated by MPInside when the application is outside of the MPI functions. If running on a patched kernel or kernel higher than 2.6.32 that allows `perf_events`, it may be of great interest to get some processor internal or PAPI counter reports not polluted by the MPI internal processing. In both cases, the usual Perfsuite output files will be created in addition to the MPInside ones. The Perfsuite outputs will have to be post-processed by `psprocess`. The way Perfsuite runs, in a such case, can be controlled by the Perfsuite environment variables. In particular, the Perfsuite configuration file used will be pointed to by the `PS_HWPC_CONFIG` environment variable. You need to ensure that the Perfsuite environment is properly installed and that the Perfsuite library is in the LD_LIBRARY_PATH list.

MPINSIDE_PCL_EVENTS : [O|A@]<PCL events list>

For system running the Linux 2.6.32 kernel or higher, CPU counters are available to end-user applications without any kernel patch or additional kernel modules. MPInside will dynamically open libfpm4.so. MPInside is not linked with this library. You need to download this library from http://perfmon2.sourceforge.net/, install it, and to set the LD_LIBRARY_PATH variable to include the install path in its list. The libfpm4.so library, written by Stephane Eranian, allows access by explicit names to numerous native counters. A list of such counters can be viewed by running the showevtinfo command included with libfpm4.so and bundled in the MPInside environment. This list of explicit counter names is far more complete than the one available with the perf command. Counting is performed only inside the MPInside window of observation. The specified counting occurs for the whole program. Counter values are displayed at the bottom of the mpinside_stats file (with post processing and displayed with a spreadsheet).

Usage Example

```
setenv MPINSIDE_PCL_EVENTS A@PERF_COUNT_HW_INSTRUCTIONS,LLC_REFERENCES
```

Note that in this example the first counter is a standard generic perf_event counter while the second is Intel Xeon Processor 5500 Series (code name Nehalem) specific.

Another example is, as follows:

```
setenv  MPINSIDE_PCL_EVENTS O@PERF_COUNT_HW_INSTRUCTIONS,LLC_REFERENCES
```

Miscellaneous Environment Variables

This section describes miscellaneous MPInside environment variables.

MPINSIDE_CALLSTACK_MAX_RECV_ENTRIES <Integer value>

Maximum number of Recv branches that the tool can manage Default: 256

MPINSIDE_CALLSTACK_MAX_SEND_ENTRIES <Integer value>

Maximum number of Send branches the tool can manage Default: 256

MPINSIDE_CALLSTACK_MAX_WAIT_ENTRIES <Integer value>

Maximum number of Wait branches the tool can manage Default: 256

MPINSIDE_CALLSTACK_SKIP

Number of ancestors the tool ignores starting from the MPI function leaf. For example, MPI SGI/IMPI MPI Fortran calls its C equivalent. There is no need to manage the Fortran calls. Setting this variable to 1 does not lose any information and can reduce the tool overhead.

MPINSIDE_COLLECTIVE_WINDWOW <MPI_collective_name>:<START>:<END>:

MPI collective calls to watch or to start or stop the tool. It starts MPInside when the watched routine reaches the counter START. It stops MPInside and the application when the watched routine reaches the counter END. If the END counter is not reached, the application will stop at MPI_Finalize.

Usage Example

```
setenv  MPINSIDE_COLLECTIVE_WINDWOW MPI_Bcast:300:4321
```

MPINSIDE_COMM_TO_WATCH <Integer value>

Communicator to watch with for the collective function selected with MPI_INSIDE_COLLECTIVE_WINDOW. Default is MPI_COMM_WORLD. You must set this communicator to a communicator number that contains all the ranks.

Special values are, as follows:

- Any communicator. If so you must set MPINSIDE_COLLECTIVE_WINDOW <collective function>:0:300000000. That is, starts at MPI_Init and ends at MPI_Finalize. This could be useful in conjunction with MPINSIDE_SHOW_W described just below. Results are unpredictable if MPINSIDE_COLLECTIVE_WINDOW is not set the way just described.

- Any communicator that is a duplication (created with MPI_Comm_dup) of MPI_COMM_WORLD

MPINSIDE_COMM_T_W <Integer value>

Identical to MPINSIDE_COMM_TO_WATCH. Deprecated, use MPINSIDE_COMM_TO_WATCH instead.

MPINSIDE_NON_STOPPING_WINDOW

By default, if a partial experiment is requested, either using the MPINSIDE_COLLECTIVE_WINDOW variable or by inserting an explicit call to mpinside_start(), mpinside_end() in the source code, the application is terminated immediately after writing the MPInside reports. If MPINSIDE_NON_STOPPING_WINDOW is set the program will continue normally up to the MPI_Finalize() call. Hangs could happen when using this variable with MPINSIDE_EVAL_SLT or MPINSIDE_MODEL. It should work fine for basics experiments or/and with MPINSIDE_EVAL_COLLECTIVE_WAIT.

MPINSIDE_SHOW_W <Integer value>

If set, a print to stdout will be done at each MPINSIDE_SHOW_W calls of the watching function. This is one way you can use to figure out which counter to set in order to select a window of observation allowing profiling the application only for some reduced meaningful steps. An example print output is, as follows:

```
Rank 0 1000 calls to MPI_Bcast with comm.=2 comm_sz=64 Elapse: 1532.004
```

MPINSIDE_CUT_OFF <real value>

Do not print branches whose time is lower than MPINSIDE_CUT_OFF percent of the total communication time. Default is 0, 01, 1%.

MPINSIDE_DELAY_AT_INIT <Integer value>

For debugging. Sleep this long (time value in seconds) in order to get time to attach some process to a debugger like gdb. Default: do not sleep

MPINSIDE_ING_COLLECTIVE_BRANCHES

Ignore collective routines from the Callstack management in order to reduce the overhead and to concentrate on `Send`/`Recv` pairs

MPINSIDE_INTERNAL_TAG_START <Integer value>

Starting tag value for MPInside exclusive usage: Default: 2**30

MPINSIDE_LIB: <MPT|IMPI|HPMPI|SCALIMPI>

MPI library used by the application. If this variable is not set, MPT is assumed.

MPINSIDE_BINARY_MATRICES_DIR Directory

Directory on to put binary matrix files. Default: `MPINSIDE_MAT_DIR`

MPINSIDE_MAT_START_STOP <start float value: start stop value>

If set, MPInside will start populating the matrices of transfer at time start and flush them at time stop and will terminate the run. The purpose of this variable is to be able to get representative matrices of transfer for input to the placement tool sergeant that skip the initialization and run few application steps. Some other ways to reduce the run with MPInside : `MPINSIDE_PARTIAL_EXPERIMENT`, `MPINSIDE_COLLECTIVE_WINDOW`, the former needing to change the source code, the latter needing to detect an MPI collective function involving all ranks that is called regularly during steps. The `MPINSIDE_MAT_START_STOP` allows shortening the run in any case. Note that the binary matrices will be the only files produced.

MPINSIDE_OUTPUT_PREFIX

Output prefix used by MPInside. Default: `mpinside` Note that this could be a full path name allowing dispatching outputs in different directories.

MPINSIDE_PARTIAL_EXPERIMENT

If set, MPInside will only start if the application calls mpinside_start() and will end either when MPI_Finalize() is called or when mpinside_end() is called. Note that the application will end as soon as mpinside_end() is called except if the variable MPINSIDE_NON_STOPPING_WINDOW is set. In this last case, the profile is frozen, the application will continue normally and the report will be written when MPI_Finalize will be called. Note also that these two calls MUST be collective calls involving all ranks. You must ensure that when calling mpinside_start() and mpinside_end() that no pending MPI requests still remain. For example, a request generated by a MPI_Irecv call before calling one of the two functions without the corresponding MPI_Wait called before calling the two functions. When building the binary, link with libMPInside_stub.so. This library must be in a directory listed in the LD_LIBRARY_PATH variable if the built binary is not run prefixed by the MPInside launcher. In such a case, the two functions above will have no effect.

MPINSIDE_PRINT_ALL_COLUMNS

Depending on the feature activated and the xxx MPI function activated some w_xxx or b_xxx columns are present in the mpinside_stats file. If this variable is set, and if xxx was activated, and if a w_xxx column or b_xxx may exist then such these columns will be reported even with full zero. Using this variable allows easier chart comparisons (same legends, same colors) between a basis run and a perfect run, for example.

MPINSIDE_PRINT_DIRTY

Print data with full precision but no formatting. With this option, the columns will look bad (not aligned) if edited with a text editor like vi. But they will be automatically well formatted again when imported into a spreadsheet.

MPINSIDE_PRINT_SIZ_IN_K

Print transfer sizes in Kbytes instead of Mbytes (the default) in the mpinside_stats file.

MPINSIDE_SHOW_READ_WRITE

Include in the `mpinside_stats` file two columns indicating the time, number of char, and number of calls to the `libc` `read()`, and `write()` functions. Note that this time is already excluded from the "`comput`" column. This is also "`comput`" time' that is, time spent outside MPI.

MPINSIDE_TRANSLATE_PERSISTENTS <Nb_entries, default 128>

Off, by default, for basic experimentations. On for `MPINSIDE_MODEL` or `MPINSIDE_EVAL_SLT`. By default, functions like `MPI_xxx_Init`, `MPI_Start`, are just executed. When on, MPInside keeps the calls that were set at the `MPI_xxx_init` calls and runs the corresponding `MPI_Ixxx` function. For example, a sequence like: `MPI_Recv_Init(buff,count,datatype,dest,tag,com,request);MPI_Start(request)` `MPI_Wait_Request` will be executed `MPI_Recv_Init(buff,count,datatype,dest,tag,com,request)` with only MPInside internal setting and then `MPI_Irecv(buff,count,datatype,dest,tag,com,request)` instead of `MPI_Start(request)` and then `MPI_Wait` (no changed). This option is **on** when `MPINSIDE_MODEL` or `MPINSIDE_EVAL_SLT` is set but can also work with basic profiling.

Usage Example

```
setenv MPINSIDE_TRANSLATE_PERSISTENTS
setenv MPINSIDE_TRANSLATE_PERSISTENTS 25
```

mpinside_stats File

This section describes the `mpinside_stats` file.

Note: The prefix name of the statistics files resulting from the MPInside command can be chosen using the `MPINSIDE_OUTPUT_PREFIX` environment variable. We are using the default prefix `mpinside` here.

Figure 1-6 on page 22 an example of the array of values printed in the `mpinside_stats` file.

>>>>	Elapse	times	in	(s)	<<<<			
CPU	Comput	w_recv	recv	w_wait	wait	send	irecv	iprobe
0	26.0021	3.6752	0.7912	0.382	0.502	7.386	0.019	3.5194
1	26.0324	3.8538	0.7169	0.5747	0.5538	7.644	0.0211	2.9284
2	26.3342	3.5806	0.6415	0.7484	0.535	7.503	0.0175	2.9643
…………								
>>>>	Mbytes	with	send	attribute	<<<<			
CPU	Comput	w_recv	recv	w_wait	wait	send	irecv	iprobe
0	------	0	0	0	0	996	0	0
……… .								
>>>>	Number	of	requests	with	Send	attribute<<<<		
CPU	Comput	w_recv	recv	w_wait	wait	send	irecv	iprobe
0	------	0	0	0	0	15008	0	0
………								
>>>>	Mbytes	with	Recv	attribute	<<<<			
CPU	Comput	w_recv	recv	w_wait	wait	send	irecv	iprobe
0	------	0	535	0	0	0	443	0
……… ..								
>>>>	Number	of	requests	with	Recv	attribute<<<<		
CPU	Comput	w_recv	recv	w_wait	wait	send	irecv	iprobe
0	------	503	503	8062	14464	0	14464	12075883

Figure 1-6 Array of Values Printed in the `mpinside_stats` File

User Counters

A set of four user counters is available allowing user time measurement to be included in the `mpinside_stats` file. Figure 1-7 on page 22 shows an example.

CPU	Comput	Hostcnt0	hostcnt1	recv	waitall	wait
0	207.650	12.483	56.987	8.119	0.004	75.605
1	210.723	14.798	45.876	1.645	0.006	74.165

Figure 1-7 `mpinside_stats` User Counters

```
C
void mpinside_host_counter (int counter, int step)

Fortran

MPINSIDE_HOST_COUNTER(counter, step)
INTEGER COUNTER, STEP

Step = -1: start counter;
Step = 1: stop counter
```

Note these counters are already accounted in the MPInside counters (comput, MPI_Recv) The number of pair requests to this function comes in the Send request set.

Columns Meaning

In order to get columns well aligned on a text file, the name of the MPI functions are shortened. The following describes such abbreviations.

b_<Collective_function>

Time spent by the MPI_Barrier inserted before the collective function if the MPINSIDE_EVAL_COLLECTIVE_WAIT environment variable was set. In this case the total time for the collective function is b_<Collective_function> + <Collective_function>. This assumes the real <Collective_function> time was not too polluted by this change. This is true in most cases.

w_<wait_or_receive_func>

Time where Sends were late in regard to they matching MPI_Recv of MPI_Wait for receive. This is equal to hour_entering_matching_send_function - hour_entering_wait_or_receive_func. If this quantity is positive, the time reported in the receive (MPI_Recv) or wait (MPI_Wait) columns is the time of the effective transfers, that is, the times taken by the receives when the sends were ready (or said another way posted prior the MPI_Wait call).

MPI_Sendrecv
MPI_Sendrecv_replace functions

In order to figure out the amount of wait time involved with these functions they are executed the following way on both sides.

MPI_Isend: reported as MPI_Sendrecv_S / MPI_Sendrecv_replace_S

MPI_Recv: reported as MPI_Sendrecv_R / MPI_Sendrecv_replace_R

MPI_Wait on the MPI_isend request : reported as MPI_Sendrecv_WS / MPI_Sendrecv_replace_WS.

The total elapsed time is the sum of the comput column and all the MPI columns (excluding the last l_recv column).

The wmxxx columns (only if MPINSIDE_PARTNER_MATCH is set to TAG or CHECKSUM) are the times from the xxx functions that cannot be matched with any matching send. This time is a part of the xxx time but not an error for it. This number allows you to figure out the accuracy of the W_xxx time reported. For a given receive, MPInside knows very precisely the time of the function.

Bytes Transferred and Number of Requests Arrays

Following the arrays reporting the elapse time for the functions are 2 set of 2 aligned arrays.

```
Set Send:
>>>> Ch_send array: Kbytes with send attribute <<<< >>>> R_send array: Number of requests with Send attribute<<<<
Set Receive:
>>>> Ch_recv array: Kbytes with Recv attribute <<<< >>>> R_recv array: Number of requests with Recv attribute<<<<
```

Point to Point Function

Point to point functions, like MPI_Send and MPI_Recv, are easily dispatched into this two logical set of arrays. A function like MPI_Wait is not of this last kind and so is arbitrarily assigned to the Recv set. The section below describes the choice taken for this kind of function. For point to point functions, the cumulated sizes reported are the true size transferred. For collective operations, things are more complicated. MPInside does its best to dispatch the more information it can on these two sets with the rules described in "Collective functions" on page 25.

Collective functions

This section describes collection functions, as follows:

- Communicator size is cumulated in the "R_send array: Number of requests with Send attribute".
- The number of calls to the collective function is always in "Ch_recv Number of requests with Recv attribute".

For collective with root (for example MPI_Bcast), as follows:

- "Ch_send array: Kbytes with Send attribute" gets the number of time the caller rank was root.
- "Ch_recv array Kbytes with Send attribute" gets the size argument.

For non rooted functions (for example MPI_Allreduce), as follows:

- "Ch_send array: Kbytes with Send attribute" gets the sendcount argument (if any) or the average if sendcounts.
- "Ch_recv array Kbytes with Send attribute" gets the size revcount argument (if any) or the average if sendcounts.

  In addition, a string at the beginning of the mpinside_stats file reports the way things are dispatched.

Here a example on eight CPU system calling MPI_Alltoall with MPI_COMM_WORLD as communicator:

```
>>> column meanings <<<<
 alltoal  :         Mpi_Alltoall :
Ch_send+=sencount,R_send+=comm_sz;Ch_recv+=recvcount,R_recv++
 send    :          MPI_Send :
 recv    :          MPI_Recv :

>>>> Ch_send array: Kbytes with send attribute <<<<
CPU   Comput   alltoal    send  recv
0000  ------        8        0    14

0007  ------        8        14    0

>>>> R_send array: Number of requests with Send attribute<<<<
0000  ------       16        0     7
```

```
0007  ------       16        7     0

>>>> Ch_recv array: Kbytes with Recv attribute <<<<
0000  ------        8        14    0

0007  ------        8        0     14

>>>> R_recv array: Number of requests with Recv attribute<<<<
0000  ------        2        7     0

0007  ------        2        0     7
```

The string "Ch_send+=sencount,R_send+=comm_sz;Ch_recv+=recvcount,R_recv++" does the following, for each call to MPI_Alltoall , the array Ch_send is incremented by the sendcount argument of the function, the R_send array is incremented by the size of the communicator (here eight), The Ch_recv array is incremented by the recvcount argument of the function and the R_recv is just incremented by 1.

The example, above, shows two calls to MPI_Alltoall with communication Size 8 = 16/2 and sizes argument were 8K. This 8K is to compare with the 14K reported for 7 calls to MPI_Sendand MPI_Recv. These two last functions are certainly not those that transfer the more bytes. As MPI_Alltoall average communicator size is 8 the amount of char transferred both sides is in the other of 8x8=64K even if last size depends on the algorithm used by the MPI library.

mpinside_stats_.M-N files

These files contain matrices with N columns and total number of CPU lines. For a run with less that 255 CPUs (say N), only one file is produced: Mpinside_clstk.0-(N-1). The file is reduced to 256 columns in order to be imported into a spread sheet.

To get such files the MPINSIDE_MATRICES variable must be set.

- **Wait Times Matrix**

The time (FWT) the CPUs are waiting for Recv. This is the time viewed by the application. It doesn't include the collectives functions. WT(I,J) = Time CPU "J" was waiting for CPU "I".

- **Send_Late Matrix**

  The Time (SLT) of the above wait because the Sender was just late. This matrix is only available with `MPINSIDE_EVAL_SLT` or `MPINSIDE_CALLSTACK_DEPTH` > 1 and `MPINSIDE_CROSS_REFERENCE`.

- **Mbytes received**

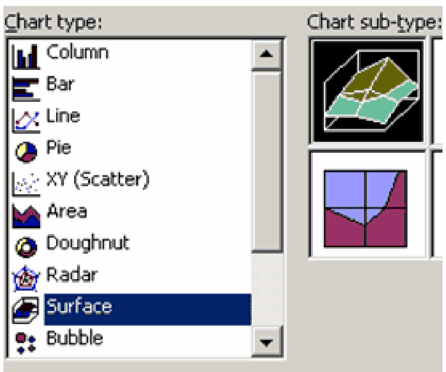
  `Mb(I,J) = Mbytes` received By CPU "J" from CPU "I".

- **Number of requests**

  Nr(I,I) = Number of requests preformed by CPU "J" With "I" as sender.

These matrices can be easily combined in the spreadsheet to provide other metrics, such as: Time to account to the MPI/hardware only = "Wait Times Matrx" – "Send_Late Matrix".

Figure 1-8 on page 28 shows the Wait time matrix chart.



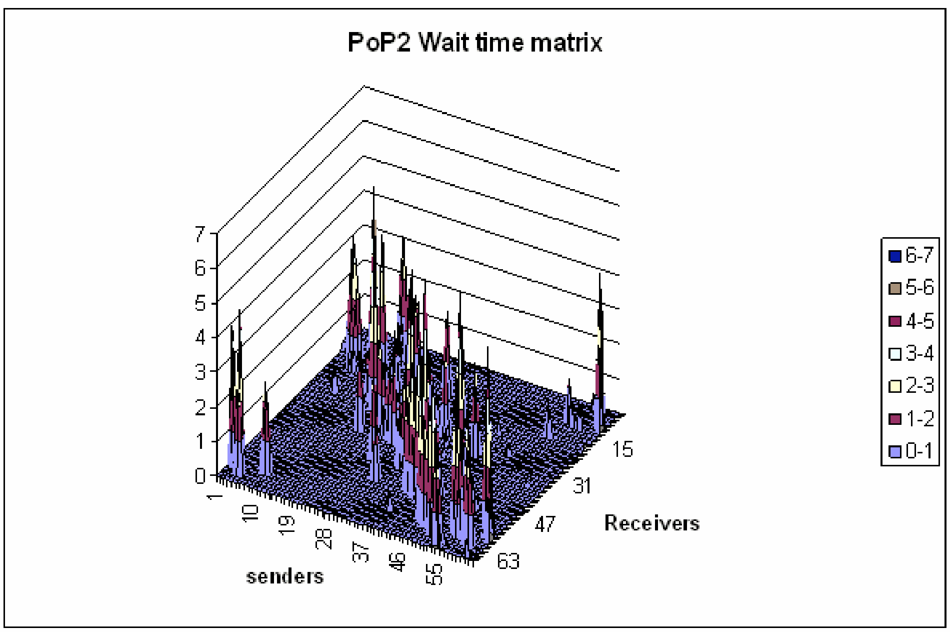


Figure 1-8 PoP2 Wait Time Matrix

Figure 1-9 on page 29 shows the MPI/hardware only matrix.

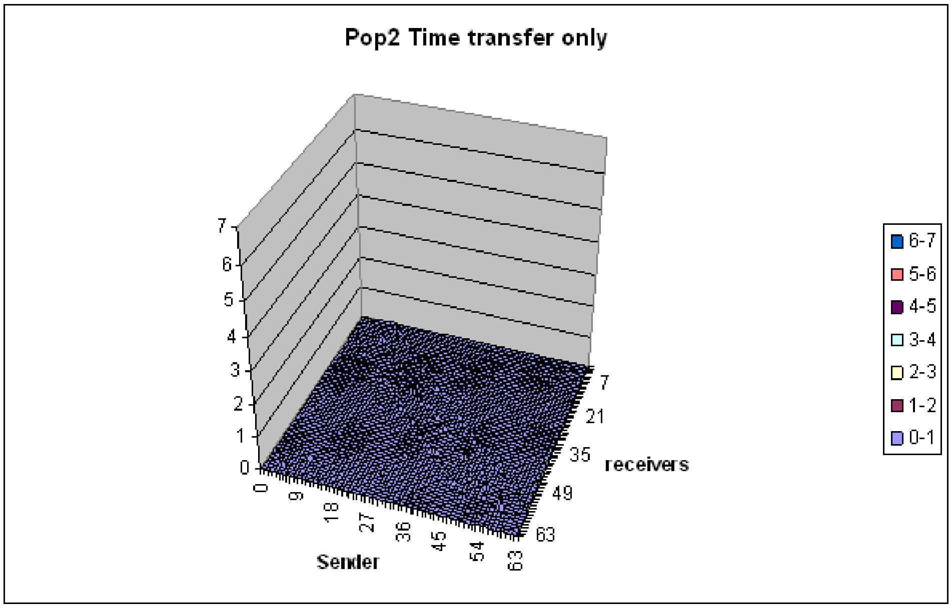


Figure 1-9 PoP2 Time Transfer Only

The following section describes the accounting of Collective operations in these files.

MPInside Binary Transfer Matrices

When the `MPINSIDE_MATRICES` value contains the "`B`" flag, binary files are written, one per rank in the `./MPINSIE_MAT_DIR` directory or elsewhere if `MPINSIDE_BINARY_MATRICES_DIR` is set. From these files, the `pram` utility, depending of the option chosen, can extract a set of ranks to build a text file equivalent to the `mpinside_stats.M-N` file described above, it can summarize the activity to the node level. But the main purpose of these files is to be input for the MPI placement tool "Sergeant". For this purpose, these files may not exactly report what is transferred but may report what is important for placement the lighter way as possible. The `pram` utility, in the following section, describes exactly what is reported.

`pram` Utility

The syntax for the `pram` utility is, as follows:

```
pram [-h] [-i] [-s] [-R][-m] [-f first_column] [-e last_column] [-o output_name] [-c cpu_list]
[-n nb_cpu_per_node] [-S nb_node_per_switch] [-k kind_of_report] [input_dir]
Print Ascii array report from Binary MPInside matrices
        -h  : this usage
        -i  : print input file information
        -s  : symetrize Matrices as Sergeant Placement tool
        -R  : reverse line order, off by default
        -m  : merge P2P and collective,default keep them separated
        -f  first_column : first column, default 0
        -e  last_column,default ncpu-1 if -k RANK,nb_nodes -1 otherwise
        -o  output_name, default mpinside_mat_frb
        -c  cpu_list,for example:$PBS_NODEFILE
        -n  nb_cpu_per_node,if -c not specified, default 8
        -k  RANK|NODE, default RANK. For example -k NODE : node to node transfer matrice
            input_dir: Directory where binary file are. Default MPINSIDE_MAT_DIR/
```

`mpinside_clstk_post.xxx` Files

Note: The prefix name of the statistics files resulting from the MPInside command can be chosen. This manual uses the default prefix "`mpinside`".

When run with the `MPINSIDE_CALLSTACK_DEPTH` variable set to a value at least 2, the MPInside tool produces a set of `mpinside_clstk.xxx` files (one per MPI ranks) containing a sorted list of branches that have to be post processed by `MPInside_post` to produce a set of `mpinside_clstk_post.xxx` readable files. A branch consists of an MPI function as leaf followed by all its "Callstack" ancestors. The branches are sorted by the measured time spent in this particular MPI function.

Each branch comes with the following statistics:

```
MPI_FUNCTION  Brid  Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS avr_szR W_miss% Rcv_W(s)
```

`Brid`	Unique identification of this branch
`#reqs_S / #reqs_R`	Number of `send`/`Recv` requests accountable to this branch

avr_szS / avr_szR	Average size sent in Mbytes accountable to this branch (meanings are depending of the MPI functions. See the descriptions for the mpinside_stats file described in "mpinside_stats File" on page 21.
W_miss%	For "Recv branches" this is the percentage of the wait time corresponding to this "Recv branches" that MPInside was not able to match with any Send Branch. (See "wm_xxx" functions in "Columns Meaning" on page 23 for more information about these misses) This value is always zero for other kind of branches.
Rcv_W(s)	The "Time(s)" column gives the time spent for the receive function itself. The "Rcv_W(s)" gives the wait time associated. For functions, such as, MPI_Recv(), these two times are equal. They are not equal for functions like MPI_Irecv().

This line is followed by the name of leaf ancestors. When called with the -l flag, the MPInside post also reports the full path of the source code and the line number of that call, provided that this information is available in the binary (compiled with the -g flag).

An example is, as follows:

```
MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
        MPI_Recv  #1214     3.923 15.94  15.9       0   66598       0     836
Ancestors: mpp_recv_real8_scalar mpp_do_update_old_r8_3d mpp_update_domain2d_r8_3d

    MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
        MPI_Recv  #1559     3.582 14.56  30.5       0     364       0  949846
Ancestors: mpp_recv_real8_scalar mpp_do_global_field2dold_r8_3d mpp_global_field2d_r8_3d

    MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
        MPI_Recv  #1226     1.719  6.99  37.5       0   11874       0    1950
Ancestors: mpp_recv_real8_scalar mpp_do_update_old_r8_3d mpp_update_domain2d_r8_3d

    MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
        MPI_Recv  #1368     1.324  5.38  42.9       0    3600       0    6016
Ancestors: mpp_recv_real8_scalar mpp_do_update_old_r8_3d mpp_update_domain2d_r8_3d

    MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
        MPI_Recv  #1491     0.880  3.58  46.5       0      66       0  112800
```

```
Ancestors :mpp_recv_real8_scalar mpp_do_update_old_r8_3d mpp_update_domain2d_r8_3d
```

The same example with with the `MPInside_post -l` flag is, as follows:

```
MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
        MPI_Recv  #1214     3.923 15.94  15.9       0   66598       0     836
Ancestors:mpp_recv_real8_scalar  /Tmpp/include/mpp_transmit.inc:168
             mpp_do_update_old_r8_3d  /Tmpp/include/mpp_do_update_old.h:338
              mpp_update_domain2d_r8_3d  /Tmpp/include/mpp_update_domains2D.h:114

    MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
        MPI_Recv  #1559     3.582 14.56  30.5       0     364       0  949846
Ancestors:  mpp_recv_real8_scalar  /Tmpp/include/mpp_transmit.inc:168
             mpp_do_global_field2dold_r8_3d  /Tmpp/include/mpp_do_global_field_old.h:146
              mpp_global_field2d_r8_3d  /Tmpp/include/mpp_global_field.h:87

    MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
        MPI_Recv  #1226     1.719  6.99  37.5       0   11874       0    1950
Ancestors:  mpp_recv_real8_scalar  /Tmpp/include/mpp_transmit.inc:168
             mpp_do_update_old_r8_3d  /Tmpp/include/mpp_do_update_old.h:338
              mpp_update_domain2d_r8_3d  /Tmpp/include/mpp_update_domains2D.h:114

    MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
        MPI_Recv  #1368     1.324  5.38  42.9       0    3600       0    6016
Ancestors:  mpp_recv_real8_scalar  /Tmpp/include/mpp_transmit.inc:168
             mpp_do_update_old_r8_3d  /Tmpp/include/mpp_do_update_old.h:338
              mpp_update_domain2d_r8_3d  /Tmpp/include/mpp_update_domains2D.h:114
```

With `MPINSIDE_CALLSTACK_DEPTH >= 2`, if the `MPINSIDE_CROSS_REFERENCE` variable is also set, some branches (depending of the MPI function leaf) have partners. Partners connect branches together. Partners are sorted by the time they induced for the MPI functions. The following is an example for a system with eight CPUs:

```
MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
        MPI_Recv  #1214     4.751 16.47  16.5       0   66598       0     836
Ancestors: mpp_recv_real8_scalar mpp_do_update_old_r8_3d mpp_update_domain2d_r8_3d
Partners_l_0:   3:#190:49.29:99.22 5:#190:18.92:99.86 2:#190:10.78:96.60 6:#194:9.28:94.30
1:#189:5.79:91.97 7:#192:5.46:86.87

...........
...........
```

```
    MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
   MPI_Allreduce  #3207     0.060  0.21  87.7      50       0       8       0
Ancestors: mpp_sum_real8_scalar mpp_global_sum_r8_2d volume_conservation
............
............
    MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
       MPI_Isend  #6     0.003  0.01  98.0     852       0     818       0
Ancestors: mpp_send_real8_scalar mpp_do_update_old_r8_3d mpp_update_domain2d_r8_3d

..........
..........
   MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
        MPI_Wait  #2448     0.001  0.00  99.4       0      88       0       0
Ancestors: mpp_sync_self get_1_from_xgrid_repro get_side1_from_xgrid
Partners_l_0:   0:#454:98.32 0:#417:1.68
```

The following is another example of a "`Recv Branch`" post-processed with the `--a --l` flags of `MPInside_post`. It shows the partner information in a different, less compact, format for easy plotting with Excel, as shown in Figure 1-10 on page 34:

```
    MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR W_miss%  Rcv_W(s)
       MPI_Irecv  #375     0.055  0.93  64.4       0   35280       0     497   0.0    0.505315
Ancestors: boundary_mp_boundary_2d_dbl_??unw  /tmp/ipo_ifortG5VfwV.f:0
            step_mod_mp_step_??unw  /tmp/ipo_ifortG5VfwV.f:0
             MAIN__  ??:0
              main  ??:0
       CPU:#brid     Self%   Wait(s) Send_lat(s)
          57:#54     79.02    0.3993    0.3906
          56:#54      7.01    0.0354    0.0051
          8:#119      6.83    0.0345    0.0000
          1:#119      4.13    0.0209    0.0157
          2:#119      1.84    0.0093    0.0070
          58:#54      0.66    0.0033    0.0005
          0:#119      0.50    0.0025    0.0000
```

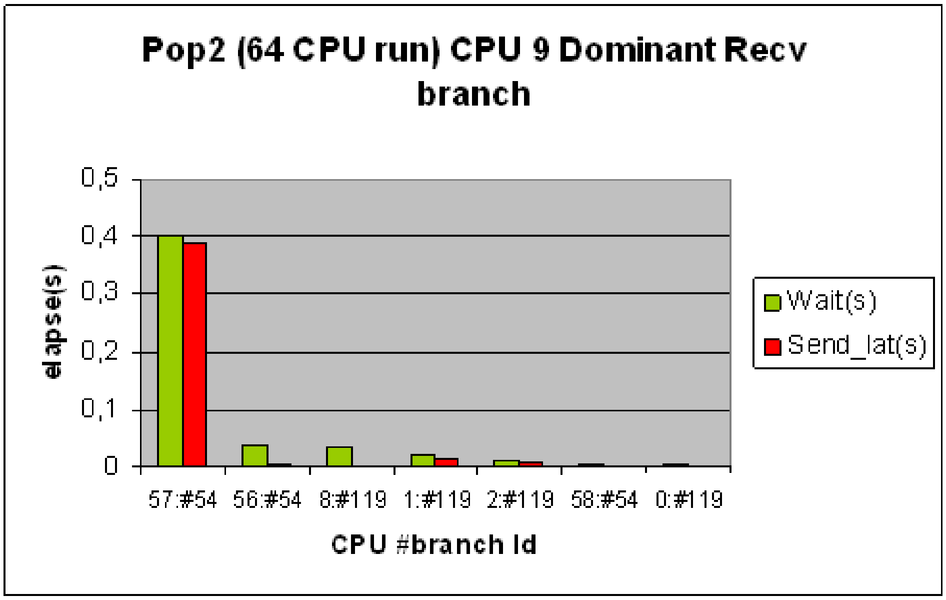


Figure 1-10 Pop2 (64 CPU run) CPU 9 Dominant Recv Branch

The partner information consists of four numbers (`Recv` branches) or 3 numbers (Wait branches) separated by ":":

A `partner` is defined by four numbers A:#B:C:D There numbers are defined, as follows:

- A

  Rank number that did the MPI `Send`/`Isend` for this branch.

- #B

  `MPI_Send`/`Isend` Branch `ident(#brid)`

- C

  Percent of this `MPI_Recv` that involved this "A" rank "#B" MPI Send branch.

- D

  Percent of this `MPI_Recv` where the corresponding `Send` was arriving late.

For example:

```
3:#190:49.29:99.22
```

This `MPI_Recv` branch id 1214 was "partner" with the rank `3` `MPI_Send` branch ID `#190` (below such branch from the `mpinside_clstk_post.3` file) and this partnership is accountable for 49.29% of this MPI_Recv branch communication time and 99.22 % of this 49.29% was just wait because the sends were arriving late.

```
  MPI_FUNCTION  Brid   Time(s) Self%  Tot% #reqs_S #reqs_R avr_szS  avr_szR
       MPI_Isend  #190     0.189  0.66  82.5   82998       0     820       0
Ancestors: mpp_send_real8_scalar mpp_do_update_old_r8_3d mpp_update_domain2d_r8_3d
```

There are four kinds of branches depending on the MPI leaf, some have partners, some do not have any partners. The interpretation of the timings of such partners also depends on the leaf kind.

Partner Branch Kinds

This section describes various CPU branch partners.

Ordinary Branches

Such branches have no partner and cannot be a target with `Recv` or `Wait` branches (see "Recv branches" on page 36 and "Wait Branches" on page 36). All collectives operations are ordinary branches.

Send Branches

Such branches have no partner but are the target of `Recv` or `Wait` branches. Leaves for this kind of branch are, as follows:

```
MPI_Ibsend
MPI_Irsend
MPI_Isend
MPI_Issend
MPI_Sendrecv_replace_S
MPI_Sendrecv_replace_WS
MPI_Rsend
```

```
MPI_Sendrecv_S
MPI_Sendrecv_WS
MPI_Send
MPI_Ssend
```

Recv branches

Such branches are targets of the `Wait` branches and have `Send` branches partners. Leaves for this kind of branch are, as follows:

- `MPI_Irecv`

  The "C" partner value means the percentage to account to this particular partnership in regard to the total `WAIT` time (The time of the `MPI_Wait` and `MPI_Waitall` that processed the `MPI_Irecv` request. This is not the time of the `MPI_Irecv` itself).

- `MPI_Recv`

  The "C" partner value means the percentage to account to this particular partnership in regard to the total `MPI_Recv` time of this branch.

- `MPI_Sendrecv_replace_R`

  `MPI_Sendrecv_R`

  Like `MPI_Recv`, these leaves are related to `MPI_Sendrecv` and `MPI_Sendrecv_replace` (see "Columns Meaning" on page 23 for how such functions are performed with MPInside)

The partners format is : A:#B:C:D is described in "Non-synchronized Send/Receive Pair Definition and Terminology" on page 2.

Wait Branches

Such branches have partners on the same processors. The partners connect the wait branches to their corresponding requests. Targets for them are "`Send` leaves" and "`Recv` leaves". The partners format is : A:#B:C, as follows:

- A

  A is always equal to `xxx` for a particular `mpinside_clstk_post.xxx` file.

- #B

  #B is always a branch that can be found in the very same mpinside_clstk_post.xxx file.

- C

  C is the percentage of time of this Wait branch that relies to the #B branch.

For wait branches that involve multiple requests (MPI_Waitall, MPI_Waitsome, and so on) this time is prorated between the corresponding branches. The following paragraphs describe how things are done on a particular MPI function basis:

MPI_Testall,
MPI_Testany,
MPI_Testsome,
MPI_Test

The time of all the MPI_testxxx functions is accounted accurately in the mpinside_stats file but the Wait time accounted to the partners only includes the last successful MPI_testxx. So the Wait time reported here has no meaning. It still allows to connect the MPI_Testxxx function to its corresponding Send or Recv branch. Such "non wait time meaning" also applies to the corresponding Recv or partners.

MPI_Waitall

Properly dispatching the timings for such function is a big issue. MPInside does not try to dispatch them. It describes, in this case, how time is dispatched between partners. In reverse, the times reported in the MPI_Waitall(waitall) column in the mpinside_stats file must correct (this is the FWT time defined in introduction). The w_wtall time must also be correct. So the partner timings are meaningless but the A:#B partner fields are correct. The timings in the matrices are incorrect. They are not correct on a CPU x CPU basis in the matrices but must be correct in the mpinside_stats files that reports all what were received.

MPI_Waitany

The request that is successful will get the wait times. This is fair if we consider that if the matching send of such request was posted sooner the wait time would have been reduced. This is not fair if we consider the other request are even late and do not get any wait times.

MPI_Waitsome

All matching sends are assumed to have come prior to this `MPI_Waitsome` posting. The 'C' wait time is prorated over the requests that have completed. Said another way, the tool gave up providing SLT information for such function.

`MPI_Wait`

`MPI_Sendrecv_WS`

`MPI_Sendrecv_replace_WS`

The wait times are accurate.

Communication "Stiffness"

Each thread maintains two counters, as follows:

- Total number of `send recv` (TNSR)
- The size of the dependency chain (SDC) by language abuse.

When a `send` occurs the SDC is incremented by one and this value is included in the message header. When a `recv` completes its SDC is incremented by one and if this value is still lower that the sending SDC, the receiver SDC gets the sender SDC.

The Stiffness is defined to be the ratio SDC/TNSR. The lower the better. The following chart (see Figure 1-11 on page 38) shows "good" communication Stiffness = 1.

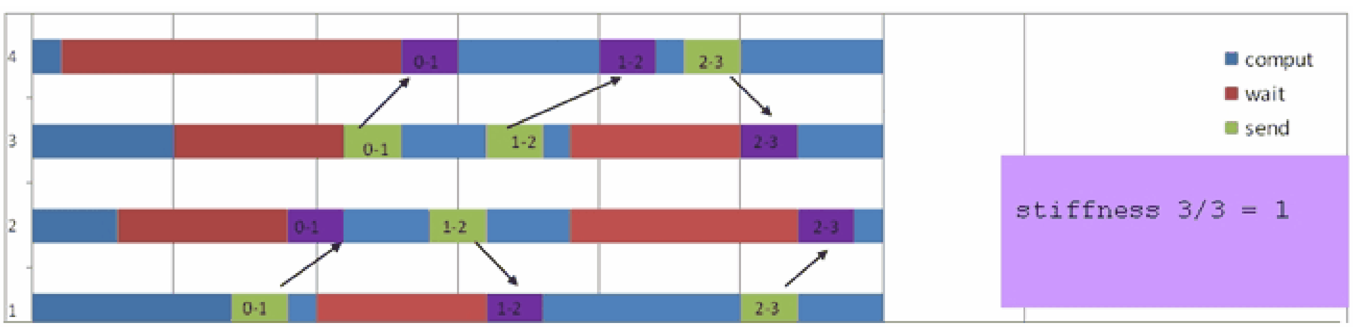


Figure 1-11 Communication Stiffness Chart 1 of 2

The following charts (see Figure 1-12 on page 39) a barrier implementation by token passing back and forth shows a "bad" communication Stiffness = 3.

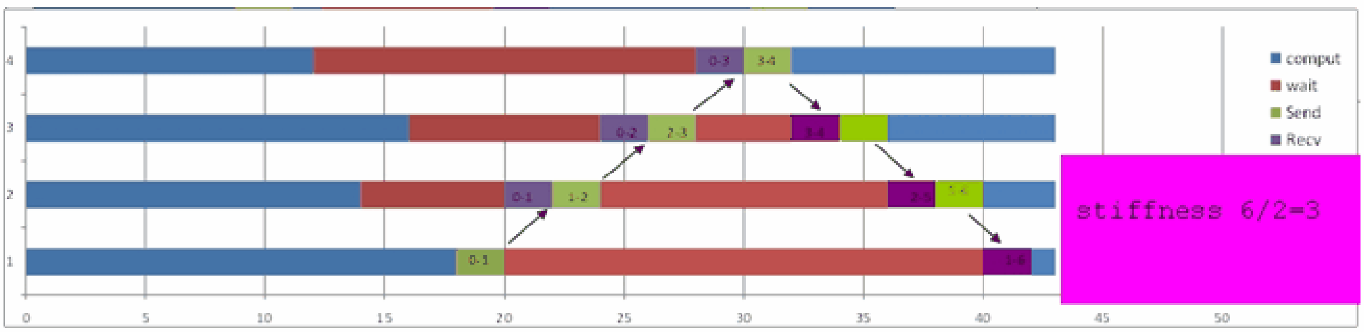


Figure 1-12 Communication Stiffness Chart 2 of 2

An application having a Stiffness growing with the number of CPU will probably have scalability issues.

Communication stiffness for the following codes are, as follows:

- PARATEC (PARAllel Total Energy Code) running on a system with 256 CPUs: Stiffness is close to 1 for all threads.
- STAR-CD running on a system with 256 CPUs:Stiffness is very different from one thread to the other between 1.5 and 15
- LINPACK running on a system with 992 CPUs: Stiffness is close to 10 for all threads

MPInside is able to give these numbers with the following environmental variable settings:

```
setenv  MPINSIDE_PARTNER_MATCH  TOKEN
setenv MPINSIDE_EVAL_SLT
setenv MPINSIDE_EVAL_COLLECTIVE_WAIT
```

In such a case : the Total number of send recv (TNSR) is reported in the "Number of request with the send attribute" array in the `mpinside_stats` file in the "Stiffness" column and the size of the dependency chain (SDC) is reported on the same column in the "Number of request with the `recv` attribute" array.

Perfect Interconnect Zero Latency Infinite Bandwidth

All the MPInside reports, described above, are also available with the communication modeled instead of being measured. In particular, MPInside is able to determine the communication value with a perfect interconnect. Knowing this asymptotic value is very useful. It can tell you if it is worth trying optimizing the application, trying another library, or spending money to acquire a machine with better MPI performance for a particular application.

To activate perfect interconnect modeling, simply run the following commands:

```
setenv MPINSIDE_MODEL PERFECT+1.0
mpirun -np xxx MPInside Your_prog Prog_args
```

All the reports previously described are available in this mode. Setting `MPINISDE_MATRICES` will create the `mpinside_stats.M-N` files and in this model mode if `MPINSIDE_MATRICES` is set to FULL, the "MB receive array" will include all the one to one communications generated by the MPT library to perform collectives operations.

Figure 1-13 on page 41 shows PARATEC running on an SGI Altix ICE system with Intel Xeon E5410 processors (code name Harpertown). The chart on the left is the result of running MPInside without any environment variable. The chart on the right is the result of running MPInside with the mentioned environment variable. All are measurements.

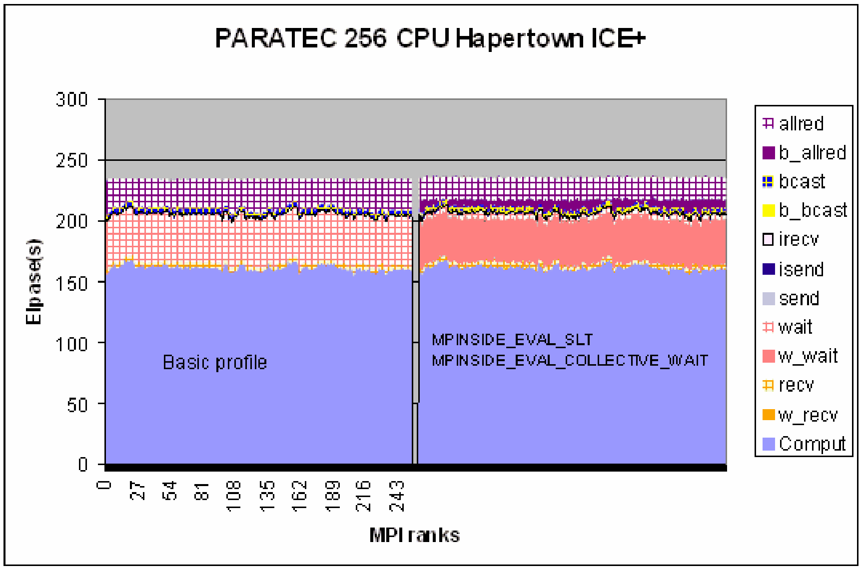


Figure 1-13 PARATEC Application on Altix ICE 1 of 2

The measurements shown in Figure 1-13 on page 41 indicate that `MPI_Allreduce` spent little time to synchronize. Most of the time is transferring (`allred` dominates `b_allred`). In reverse, there is very little time spent transferring in `MPI_Wait` (`w_wait dominates wait`). What is responsible for all this `Send` late time (`w_wait`). Is it the application itself that introduced load unbalancing or the interconnect ? The following chart that is the result of a perfect interconnect run gives us the answer (see Figure 1-14 on page 42). The application itself does not carry much load unbalance. This load unbalance depends on the interconnect performance.

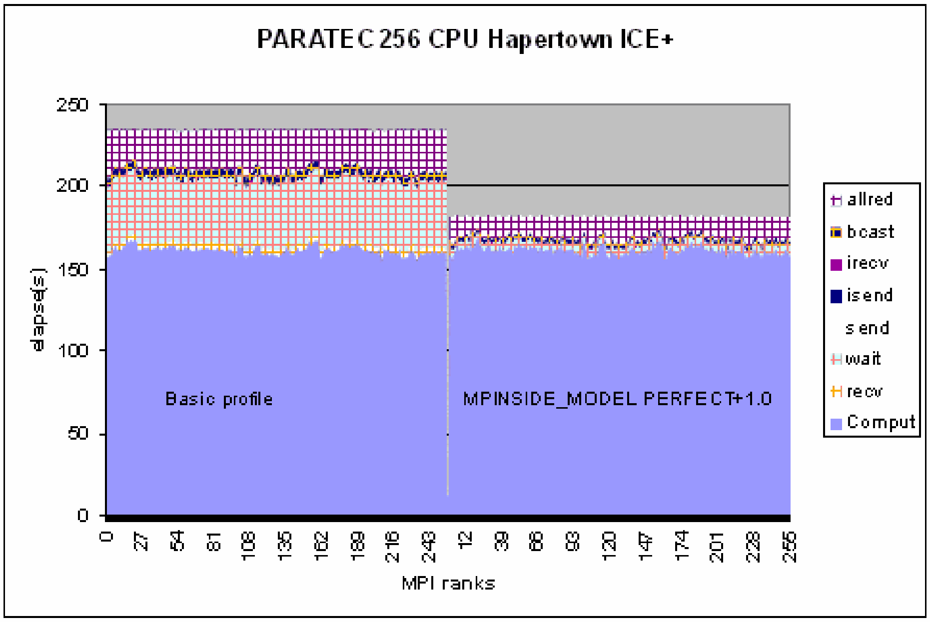


Figure 1-14 PARATEC Application on Altix ICE 2 of 2

This high load unbalance, due to the interconnect, does not happen with LINPACK on an SGI Altix ICE 8200 or an SGI Altix ICE 8400 system with 992 processors (see Figure 1-15 on page 43). On both machines, most of the communication time is from the application. The `Wait` time is not highly reduced with the perfect interconnect. Note the non-negligible `MPI_Send` (send) time. This time is not related to the interconnect but to the fact that, because of the big transfer sizes used, the sends are not buffered so the `MPI_Send` only complete when the received are done. The perfect interconnect assumes infinite bandwidth and zero latency so the `MPI_Send` (send) time disappears.

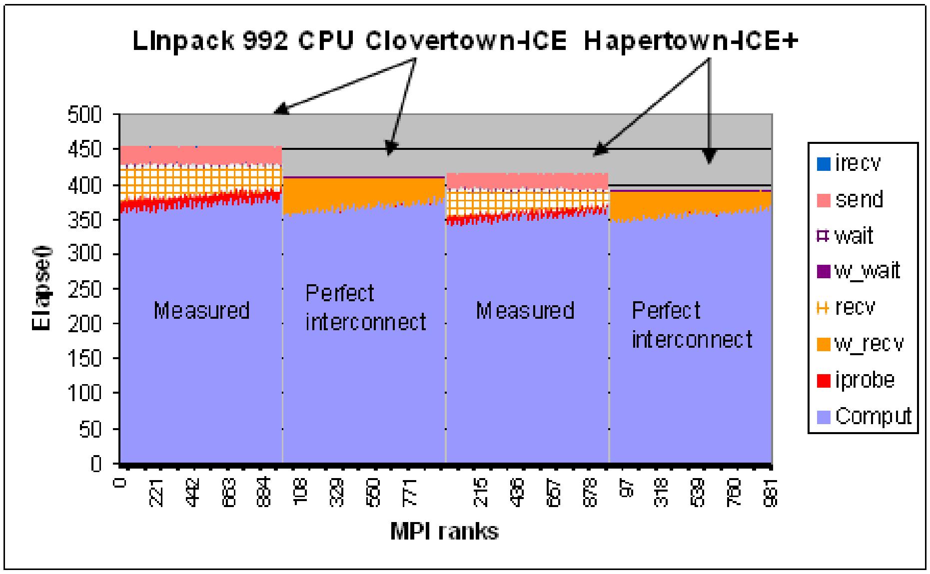


Figure 1-15 LINPACK Measure versus Perfect Interconnect Timing

Using the MPInside Profiling Tool

This chapter describes how to select a window of observation using the MPInside profiling tool.

Selecting a Window of Observation using Re-compilation

This section describes how to select a window of observation using re-compilation.

With re-compilation

```
C:
(void) mpinside_start();
(void) mpinside_end();

Fortran:
Call mpinside_start
Call mpinside_end()

They must be calls involving ALL ranks
The application terminates when mpinside_end() is encountered
If MPI_Finalize() is called prior to  mpinside_end() the application
will end normally with statistics from mpinside_start to MPI_finalize

Link
$(LD) ... -L MPInside_install_dir/lib -lMPInside_stub
```

```
Execution:

setenv LD_LIBRARY_PATH MPInside_install_dir/lib:${LD_LIBRARY_PATH}
Without the MPInside launcher the application will run normally. The calls to
mpinside_start(), mpinside_end() will have no effect.


Set MPInside options as usual
Tell MPInside to wait for the mpinside_start call to gather statistics
setenv MPINSIDE_PARTIAL_EXPERIMENT
Run MPInside as usual
mpirun -np xx MPInside your_apps [your args]
```

Selecting a Window of Observation with a Collective Function Heartbeat

This section describes how to select a window of observation with a collective function heartbeat.

```
setenv MPINSIDE_COLLECTIVE_WINDOW <Collective function>:<# call to start>:<#call to end>
Examples:
setenv MPINSIDE_COLLECTIVE_WINDOW MPI_Allreduce:1000:1300
setenv MPINSIDE_COLLECTIVE_WINDOW  MPI_Barrier:0:2000000005

setenv MPINSIDE_COMM_TO_WATCH <comunicator to watch>
special values: default :MPI_COMM_WORLD,any communicator:-1,any MPI_COMM_WORLD duplication : -2
Examples:
setenv MPINSIDE_COMM_TO_WATCH 2
setenv MPINSIDE_COMM_TO_WATCH 114685088
setenv MPINSIDE_COMM_TO_WATCH -2
Run MPInside as usual

If <#call to start>W is 0 collecting statistics starts with MPI_Init()
If MPI_Finalize is called before the <#call to end> the application
will terminate normally with the MPInside report.

If application is built with mpinside_start(), mpinside_end():
 if MPINSIDE_PARTIAL_EXPERIMENT is not set
   The two above variables (MPINSIDE_COLLECTIVE_WINDOW and MPINSIDE_COMM_TO_WATCH)
will be honored and the calls to mpinside_start(), mpinside_end() will have no other effect
but a warning message will be printed in stderr

else
    The application will abort at the MPI_Init() time
```

Using such feature only works if, as follows:

- At least one collective function is called regularly enough with a communicator or some communicators involving ALL ranks. True in most cases, but not always.

Spy the Collective Functions

Selecting the collective function and the communicator can be an issue. In order to help this selection, MPInside provides some ways to spy the collective functions.

To spy the collective functions, perform the following:

- Obtain a basic MPInside report, in order to see how often the collective functions are called.
- Select a collective function that is often called by the same number of CPUs. Such information is available in the "`Number of calls with the receive attribute`" array inside the `mpinside_stats` file.
- Assume, for example, that the `MPI_Allreduce` function looks like a good candidate. Obtain an MPInside run with the following:
  - Watch from start to end:

    ```
    setenv MPINSIDE_COLLECTIVE_WINDOW MPI_Allreduce:0:1000000000
    ```

  - Watch any communicator duplication of `MPI_COMM_WORLD`:

    ```
    setenv MPINSIDE_COMM_TO_WATCH -2
    ```

  - Have a print on `stdout` for each 100 calls to the watched collective function:

    ```
    setenv MPNSIDE_SHOW_W 100
    ```

Note that the application is run two times. The first report is useful. You could run the application only once if you knew the collective function to watch. You can check if the application actually calls some Collective MPI functions by running the following command:

```
nm executable_file |grep MPI_
```

An stdout example with POP2 using Intel MPI is, as follows:

```
------------------------------------------------------------------------
 End of initialization
------------------------------------------------------------------------
rank 0 13300 calls to MPI_Allreduce  with comm= 1140850688 comm_sz=256 elapse       45.022756
Step number  :      100
Date         : 02 jan 0000
Hour         :       18
Minute       :        6
Second       :       33
Time(days)   :       1.754545
 Rank 0 13400 calls to MPI_Allreduce  with comm= 1140850688 comm_size=256 elapse       45.512763
 Rank 0 13500 calls to MPI_Allreduce  with comm= 1140850688 comm_size=256 elapse       45.566929
......
Step number  :      200
Date         : 04 jan 0000
Hour         :       12
Minute       :       13
Second       :        5
Time(days)   :       3.509091
 Rank 0 24800 calls to MPI_Allreduce  with comm= 1140850688 comm_size=256 elapse       86.473298
```

Because of the strong "Step" meaning and by correlating the elapsed time between Step number 13400 and 24800 with the global time, it is clear that the MPI profile can be captured between MPI_Allreduce 13400 and 24800. This reduces the run to few seconds instead of hours, with these setting:

```
setenv MPINSIDE_COLLECTIVE_WINDOW MPI_Allreduce:13400:24800
setenv MPINSIDE_COMM_TO_WATCH -2
#setenv MPINSIDE_SHOW_W 100
```

MPInside(3) Man Page

This appendix provides a copy of the MPInside(3) man page for your convenience.

To see the MPInside man page online, make sure the MPInside/3.3 module is loaded, as follows:

```
uv44-sys:~ # module avail

------------------------ /usr/share/modules/modulefiles ------------------------
MPInside/3.3        module-info         null                sgi-upc/1.05
chkfeature          modules             perfboost           sgi-upc-devel/1.05
dot                 mpiplace/1.01       perfcatcher         use.own
module-cvs          mpt/2.04            scotch/5.1.11
uv44-sys:~ # module load MPInside/3.3
```

The see a copy of the MPInside(3) man page online, perform the following:

```
uv44-sys:~ # man MPInside
MPInside(3)                                                                 MPInside(3)



NAME
       MPInside -  Performance MPI data collection tool

DESCRIPTION
       MPInside is simply invoked by prefixing the unmodified MPI application executable by MPInside.
       Features are triggered using environment variables. For example:

       mpirun -np 128 MPInside apps apps_arg

       MPInside by default (with no environment variables set) creates at least a file named mpinside_stats.
       This file contains 5 sets of columns which can be easily exploited by a spreadsheet like Excel:

       Set 1     : Time outside MPI + all the MPI functions timing

       Set  2-3 : Named Ch_send-R_send, Amount of char transmitted + number of requests with the Send attribute

       Set  4-5 : Named ch_recv-R_recv,Same as Set 2-3 but with the Recv attribute.
```

Dispaching sizes and requests for point to point operation is natural.
Things are more complicated for MPI collective functions.
A string explaining what is actually cumulated in the 2-5 sets is printed on top of the mpinside_stats file. Here an example for MPI_Alltotall:

```
MPI_Alltoall: Ch_send+=sendcount,R_send+=comm_sz;Ch_recv+=recvcount,R_recv++
```

It says: Ch_send is incremented with the specified send count, R_send is incremented with the size of the MPI communicator, Ch_recv is incremented with the passed receive count and R_recv is just incremented.

MPInside can also be activated by using the LD_PRELOAD facility directly. For example:

```
setenv LD_PRELOAD /opt/sgi/mpinside/lib/libMPInside_mpt.so

mpirun -np 128 MPInside apps apps_arg
```

To coexist with libFFIO.so this last mechanism must be used to activate MPInside and the MPInside library must appear before the FFIO one. For example:

```
setenv LD_PRELOAD /opt/sgi/mpinside/lib/libMPInside_mpt.so:/usr/lib64/libFFIO.so
```

ENVIRONMENT

MPINSIDE_EVAL_COLLECTIVE_WAIT

if set, MPInside will put an MPI_Barrier (and will time it) before any MPI collective operation.
This assumes that the time of a collective operation is the time of all processors to synchronize + the time of the operation. This is not always true but it is true most of the cases and the time to really perform the collective operation is very short compared to the synchronization time.
In the mpnside_stats file, the column "b_xxx" will give the MPI_barrier time of the corresponding "xxx" MPI collective function and "xxx" the remaining time. When MPINSIDE_PARTNER_MATCH is set to TOKEN, setting MPINSIDE_EVAL_COLLECTIVE_WAIT will also lead to evaluate the "Stiffness" of the application (see below)

MPINSIDE_EVAL_SLT

If set, MPInside will measure the time the Sends were late (SLT) compared to the Recv-Wait arrivals.
Such time will be w_xxx in the mpnside_stats file. xxx could for example be MPI_Wait or MPI_Recv.

```
        It cannot MPI_Irecv,  because the Send late time, if any, will be, for this last function,
        accounted in an MPI_Wait-like function. MPINSIDE_EVAL_SLT is equivalent
        to MPINSIDE_CALLSTACK_DETH = 1  +  MPINSIDE_CROSS_REFERENCE
        except no mpinside_clstk.xxx files will be created.

MPINSIDE_WAIT_TIME_NO_CROSSREF
        deprecated, use MPINSIDE_EVAL_SLT instead.

MPINSIDE_CALLSTACK_DEPTH        <integer number>
        If  set,  MPInside  will unwind the stack up to the depth specified and a set
        of mpinside_clstk.xxx files will be created (one per rank).
        These files will contain statistics about all the branches (see definition
        above)  that  have  an  MPI function as leaf. The mpinside_clstk.xxx files
        only contain raw addresses. The address-Routine name matching is performed by MPInside_post
        that  produces  mpinside_clstk_post.xxx  files
        (see  more  information  about  the  format of such files below).
        If MPInside_post is run with the "<96>l" flag, the source file line numbers
        will be also printed (provided the application was  compiled  with  the <96>g flag).
        Note that most of the overhead of the tool is imputable to unwind the stack.
        One should take care not to set this variable to a number bigger than necessary.

MPINSIDE_CROSS_REFERENCE
        If set, MPInside instruments the Branches with "partners" providing timed
        cross CPU branches information.(
        See mpinside_clsth_post below)

MPINSIDE_LITE
        The  MPInside  overhead is very low. Nevertheless with applications
        doing a lot of calls to functions like MPI_Test, MPIprobe.....,
        it may happen the MPInside overhead accounts.
        With this variable set the overhead is  reduced to minimum.
        In such case only the timings
        will be reported in the mpinside_stats file. No size and request information will be printed
        and the only MPInside variable recognized  will  be  MPINSIDE_OUT-
        PUT_PREFIX,  MPINSIDE_VERBOSE,  MPINSIDE_NON_STOPPING_WINDOW,  MPINSIDE_SHOW_READ_WRITE,
        MPINSIDE_PARTIAL EXPERIMENT.

MPINSIDE_TRANSLATE_PERSISTENTS <Nb_entries, default 128>
        Off by default for Basic experimentations. On for MPINSIDE_MODEL or MPINSIDE_EVAL_SLT.
        By  default  functions  like  MPI_xxx_Init,  MPI_Start,  are  just  executed.
        When  On MPInside keeps what were set at the
```

```
        MPI_xxx_init  calls  and  run  the  corresponding MPI_Ixxx  function.
        For  example  a   sequence   like:
        MPI_Recv_Init(buff,count,datatype,dest,tag,com,request);
        MPI_Start(request);  MPI_Wait_Request will be executed
        MPI_Recv_Init(buff,count,datatype,dest,tag,com,request)
        with only MPInside internal setting and then
        MPI_Irecv(buff,count,datatype,dest,tag,com,request) instead of MPI_Start(request)
        and then MPI_Wait (no changed). This option is On
        when MPINSIDE_MODEL or MPINSIDE_EVAL_SLT
        is set but can also work  with  basic profiling.
        Usage  examples:  setenv MPINSIDE_TRANSLATE_PERSISTENTS.
        Setenv MPINSIDE_TRANSLATE_PERSISTENTS 256.


MPINSIDE_MATRICES   [EXA | PLA | P2P:[+|-B][S|M]
        Print transfer matrices files. Default is not to
        print any matrices files. Option:

None: Only point to point operation will be reported. (See mpinside_stats.M_N)
below for the format of the output files)

EXA     : Matrices will include exact P2P transfers implied by Collective functions. (MPT only)

PLA      :  Matrices  will include generic P2P transfers implied by Collective functions.
This is the best choice for thses matrice to be input to an automatic placement tool

+B      : In addition to the mpinside_stats.M-N. The transfer matrices size
and request will be print  in  binary format  to  be  used  as  input  for
the placement tool Sergeant. There will be one file per rank (see the MPIN-
SIDE_BINARY_MATRICES_DIR  below and the MPInside binary transfer matrices section).

-B      :Binary files will be the only ones produced

S       : Collectives and P2P matrices are separated in the binary files

M       :  Collectives and P2P matrices are merged in the binary files

Usage example: setenv MPINSIDE_MATRICES PLA:-B:S.

MPINSIDE_SIZE_DISTRI [T+]nb_bars[:first-last]
        An histogram of the request sizes distribution will be printed at
```

```
        the end of mpinside_stats for rank first
        to  last:  Default  12:0-0 (only rank zero and bar size : 0, 128, 256, 512,.....,65536.
        The cumulus of the calls for all rank is then terminated the report.
        This cumulus is always preinted even if the variable  is
        not set. If  T+ is specified each histogram of the request sizes if followed
        by a size distribution time histogram. On such histogram the time taken by function
        like MPI_Wait, MPI_Waitall. is  not  accounted  to
        these functions but to the MPI_Isend, MPI_Irecv,.. functions
        that generated the request passed to the Wait
        function.  Usage exemple; setenv MPINSIDE_SIZE_DISTRI T+120:-16
MPINSIDE_WITH_PERFSUITE  : ALL|OUT (x86 only)
        If set to ALL, the Perfsuite profiler will be activated
        concurrently to the MPInside process
        for the  execution  Window. If set to OUT the Perfsuite profiler will be activated
        by MPInside when the application is outside of the MPI functions.
        If running on a patched kernel or or kernel higher than 2.6.32  that  allows
        perf_events,  it  may be of great interest to get some processor internal
        or PAPI counter reports not polluted by the MPI internal processing In both cases,
        the usual Perfsuite output files will  be  created
        in addition  to the MPInside ones. The Perfsuite outputs will have to
        be post-processed by psprocess. The way Perfsuite will run in such case can be controlled
        by the Perfsuite env. variables In particular, the Perfsuite  .config file. used
        will be pointed by  the PS_HWPC_CONFIG env. variable. This is the user responsi-
        bility to ensure the Perfsuite eneivronment is properly installed and
        that the Perfsuite  library  are  in the LD_LIBRARY_PATH list.

MPINSIDE_PERFSUITE_OUTSIDE_MPI
        (x86 only) This variable is deprecated use MPINSIDE_WITH_PERFSUITE instead

MPINSIDE_SHOW_LATE_RECV
        This  is not available is MPINSIDE_PARTNER_MATCH is set to TOKEN.
        It is available if this last variable is set to TAG or CHECKSUM
        or MPINSIDE_MODEL is set and MPINSIDE_EVAL_SLT or
        (MPINSIDE_CALLSTACK_DEPTH  >=  1 and  MPINSIDE_CROSS_REFERENCE).
        If MPINSIDE_SHOW_LATE_RECV is set the last column "l_recv" will report
        the total time the MPI_Recv or the MPI_Wait for receive functions were posted
        ahead their matching  sends.
        If the  MPI_Send  are  properly buffered their times are very low.
        In case the MPI_Send ("send" column in the mpinside_stats file) is high
        it could be interested to compare this MPI_Send time to the time the Received
        were late reported in the l_recv column.
```

```
MPINSIDE_PCL_EVENTS : [O|A@]<PCL events list>
       For system running 2.6.32 kernel or higher,
       CPU counters are available to user
       without any kernel patch or additional kernel modules.
       MPInside is linked with libfpm4 written
       by Stephane Eranian that allows access
       by  explicit names to numerous native counters.
       A list of such counters can be viewed by running the show-
       evtinfo command coming with libfpm4
       and bundled in the MPInside environment.
       This list of explicit counter names is  far
       more  complete than the one available
       with the perf command . Counting is performed only
       inside the MPInside window of observation.
       The <event list> is a list of events
       separated by .,..  If   O@ is  set  counting
       occurs outside MPI only.
       if A@ or just an <event list> is specified
       counting occurs for the whole program.
       Counter values are displayed at the bottom of
       the mpinside_stats file  (with  still  in
       mind  a  post processing with a spreadsheet).
       Examples: setenv MPINSIDE_PCL_EVENTS .PERF_COUNT_HW_INSTRUCTIONS,LLC_REFERENCES..
       Note  in this example the first counter
       is a standard  generic  perf_event  counter
       while    the   second   is   Nehalem   specific.
       Another   example:   setenv    MPINSIDE_PCL_EVENTS   O@
       PERF_COUNT_HW_INSTRUCTIONS,LLC_REFERENCES.


Miscellaneous environment variables
       MPINSIDE_CALLSTACK_MAX_RECV_ENTRIES <Integer value>
              Maximum number of Recv branches the tool can manage (default : 256)

       MPINSIDE_CALLSTACK_MAX_SEND_ENTRIES <Integer value>
              Maximum number of  Send branches the tool can manage (default : 256)

       MPINSIDE_CALLSTACK_MAX_WAIT_ENTRIES <Integer value>
              Maximum number of Wait branches the tool can manage (default : 256)
               <Integer value> Maximum number of Recv branches the tool can manage (default : 256)
```

```
MPINSIDE_CALLSTACK_MAX_SEND_ENTRIES <Integer value>
       Maximum number of  Send branches the tool can manage (default : 256)

MPINSIDE_CALLSTACK_MAX_WAIT_ENTRIES <Integer value>
       Maximum number of Wait branches the tool can manage (default : 256)

MPINSIDE_CALLSTACK_SKIP
       number of ancestors the tool ignores starting from the MPI function leaf.
       For  example  MPI  SGI/IMPI  MPI Fortran  calls  its C equivalent.
       There is no need to manage the Fortran calls. In such case, setting this
       variable to 1 won'tlose any information and can reduce the tool overhead.

MPINSIDE_CHECKSUM_MATCH <int_to_check>
       deprecated. Use MPINSIDE_PARTNER_MATCH instead.
MPINSIDE_CLOCK_IS_SYNCRHO
       This can be used on machine having a synchronized clock
       like the IA64 Altix  single  image.
       Otherwise  if MPINSIDE_PATTERN_MATCH is set to TAG or CHECKSUM,
       MPInside must built translation tables and calibration
       tables to maintain a correlation between the different clock sources.
       These are not built without incertitude so it is best not to use such tables
       if a synchronized clock is available.

MPINSIDE_COLLECTIVE_WINDWOW <MPI_collective_name>:<START>:<END>
       MPI  collective  calls  to  watch  or to start/stop the tool.
       It starts MPInside when the watched routine
       reaches the counter START. It stops MPInside AND the application
       when the  watched  routine  reaches  the
       counter  END  except if the variable MPINSIDE_NON_STOPPING_WINDOW is set.
       In this last case the profile is written but the application
       will continue normally.
       If the END counter is  not  reached, the  application
       will stop at MPI_Finalize.
       Example : setenv  MPINSIDE_COLLECTIVE_WINDWOW MPI_Bcast:300:4321
       calls  to  watch  or  to  start/stop  the  tool.
       It starts MPInside when the watched routine reaches the counter START.
       It stops MPInside AND the application when
       the watched routine reaches the counter END.
       If the END counter is not reached, the application will stop at MPI_Finalize.
       Example : setenv  MPINSIDE_COLLECTIVE_WINDWOW MPI_Bcast:300:4321
```

```
MPINSIDE_COMM_TO_WATCH   <Integer value>
       Communicator to watch with for the collective function selected
       with MPI_INSIDE_COLLECTIVE_WINDOW. Default is  MPI_COMM_WORLD.
       You must set this communicator to a communicator number that contains all the ranks.
       Special values:

       -1: Any communicator. If so you  must  set
       MPINSIDE_COLLECTIVE_WINDOW  <collective  function>:0:300000000.
       I.e. starts  at  MPI_Init
       and ends at MPI_Finalize. This could be useful in conjunction with MPINSIDE_SHOW_W described
       just below. Results are unpredictable if MPINSIDE_COLLECTIVE_WINDOW
       is not set the way just described.
       -2: Any communicator that is a duplication (created with MPI_Comm_dup) of MPI_COMM_WORLD.

MPINSIDE_SHOW_W <Integer values>
       If set, a print to  stdout will be done at each

MPINSIDE_COMM_T_W       <Integer value>         : Identical to MPINSIDE_COMM_TO_WATCH.
      Deprecated, use MPINSIDE_COMM_TO_WATCH instead.

      If set, a print to  stdout will be done at each MPINSIDE_SHOW_W calls
      of the watching function.
      One can this way figure out which counter to set in  order
      to  select  a  window  of  observation allowing profiling
      the application only for some reduced meaningful steps. Example of such print:

       Rank 0 1000 calls to MPI_Bcast with comm.=2 comm_sz=64 Elapse: 1532.004
=     MPINSIDE_CROSS_PARTNER_STACK_SIZE <integer value>
       Only meaningful if MPINSIDE_PARTNER_MATCH is set to TAG or CHECKSUM.
       In order to provide partner  informa-
       tion,  Mpinside  must  force  the sending CPU to send some information
       to the receiving one for any calls.
       Such exchanges are stacked to reduce the overhead
      (same time to send/recv 0 bytes or 64 bytes). Note  that
       such  supplemental  messages  are  sent/received perfectly synchronized.
       By this last sentence we mean the
       reception of the supplemental message occurs at a moment
       where we are  sure  the  matching  send  is  done
       (default : 64)
```

```
 MPINSIDE_CUT_OFF <real value>
        Do  not  print  branches  whose  time  is  lower  than
        MPINSIDE_CUT_OFF  % of the total communication time
        (default is 0,01, 1%)

 MPINSIDE_DELAY_AT_INIT integer value>
        For debugging. Sleep this long (time value in seconds) in order to get time
        to attach some  process  to  a
        debugger like gdb. (default : do not sleep)

 MPINSIDE_ING_COLLECTIVE_BRANCHES
        Ignore  collective  routines  from the Callstack management
        in order to reduce the overhead and to concen-
        trate on Send/Recv pairs
MPINSIDE_INTERNAL_TAG_START <integer value>
        Starting tag value for MPInside exclusive usage: default : 2**30

 MPINSIDE_LIB: <MPT|IMPI|HPMPI|SCALIMPI>
        MPI library used by the application. If this variable is not set MPT is assumed

 MPINSIDE_BINARY_MATRICES_DIR Directory:
        directory on to put binary matrix files. Default : MPINSIDE_MAT_DIR.  directory on
        to  put  binary  matrix files. Default : MPINSIDE_MAT_DIR.
        The pram utility allows to convert such binary matrices to ascii format
        suitable for spreadsheets. This utility also allows to reduce
        the rank-to-rank  matrices  to  node-to-node
        matrices.  To  help  the visualization of big matrices an utility:
        mpinside2wrl that converts the MPInside
        matrices to vrlm format is also provided.
        The files created could  then  be  visualized  by  a  tool  like
        vrlmview that is freely available on the internet

 MPINSIDE_MAT_START_STOP  <start float value: start stop value>
        If  set  MPInside will start populating the matrices of transfer
        at time start and flush them at time stop
        and will terminate the run. The purpose of this variable is to be able
        to get representative  matrices  of
        transfer  for  input  to  the placement tool sergeant
        that skip the initialization and run few application
        steps. Some other ways to reduce the run
        with  MPInside  :  MPINSIDE_PARTIAL_EXPERIMENT,  MPINSIDE_COLLECTIVE_WINDOW,
```

```
        the former needing to change the source code, the latter needing
        to detect an MPI collective
        function involving all ranks that is called regularly during  steps.
        The  MPINSIDE_MAT_START_STOP  allows
        shortening the run in any case. Note the Binary matrices will be the only files produced.

MPINSIDE_OUTPUT_PREFIX  <file path prefix>
        Output  prefix used by MPInside. (Default mpinside.
        Note this could be a full path name allowing dispatch-
        ing outputs in different directories.

MPINSIDE_PARTNER_MATCH     <TOKEN | TOKENRISK | TAG | CHECKSUM:int>

TOKEN : This is the default. It works with a idea similar to  the  one  we  use
with  the  MPINSIDE_EVAL_COLLECTIVE_WAIT  feature. The Send late time is evaluated
by sending a zero size message (actually a 3 integers message
but this doesn<92>t take longer time than a zero size message)
prior to the "Data message". The time  to  receive
this  zero  message  minus  a  time calibrated by MPInside is the time the send
was late (the one reported in the w_xxx columns). The time reported in the xxx column is
the true time for a receive when the  sending  message  is ready.

TOKENRISK  :  With  TOKEN the MPI_Recv time (recv) is always accurate.
In revenge the other w_xxx columns (w_wait
for example) may be biased. This is because there is a risk of deadlock in some situation

TAG : MPInside matches Send/Recv  according to the transfer MPI Tag.

CHECKSUM:<int number> : If the application doesn<92>t check received requests
in the order they were  sent  (that
is  perfectly standard) and uses identical MPI Tags, the TAG option way
for matching messages may fail. If CHECKSUM is set the Send/recv matching
is based on the xor of  the  <int_to_check>  first  integer  of  the  Send/recv
buffers.  Use  this  heuristic  with application using identical tags:
Except when running on a IA64 Altix single
image machine that have a synchronized clock the TAG or CHECKSUM option must be
used  in  conjunction  with  the
MPINSIDE_INIT_CAL and MPINSIDE_SYNCHRO_CLOCK variable.

MPINSIDE_PARTIAL_EXPERIMENT
        If  set  MPInside  will  only  start  if  the  application calls mpinside_start()
```

```
and will end either when
MPI_Finalize() is called or when mpinside_end() is called.
Note the application will end as soon as  mpin-
side_end()  is  called  except  if the variable MPINSIDE_NON_STOPPING_WINDOW is set.
In this last case the
profile is written but the application will continue normally.
Note also that these two calls MUST be col-
lective  calls  involving all ranks. When building the binary,
link with libMPInside_stub.so. This library
must be in a directory listed in the LD_LIBRARY_PATH variable if
the built binary is not prefixed  by  the
MPInside launcher. In such a case the 2 functions above will have no effect.
```

```
MPINSIDE_PRINT_ALL_COLUMNS
       Depending  of  the feature activated and the xxx MPI function activated
       some w_xxx or b_xxx columns
       are present in the mpinside_stats file. If this variable is set if xxx  was activated
       and if a w_xxx column or b_xxx may exist then such these columns
       will be reported even with full zero.
       Using this variable allows easier chart comparisons(same legends, same colors)
       between a basis run and a  perfect run for example.
       Print data with full precision but no formatting. With this option the columns
       will look bad (not aligned)
       if edited with a text editor like "vi". But they will be automatically
       well formatted again when  imported into a Spreadsheet.

MPINSIDE_PRINT_SIZ_IN_K
       Print transfer sizes in Kbytes instead of Mbytes  (the default) in the mpinside_stats file.

MPINSIDE_SHOW_READ_WRITE
       Include in the mpinside_stats file two columns indicating the time, number of char,
       and number of calls to
       the libc read(), write(.TP MPINSIDE_PRINT_DIRTY Print data with full precision
       but  no  formatting.  With this  option the columns will look bad (not aligned) if edited
       with a text editor like "vi". But they will
       be automatically well formatted again when imported into a Spreadsheet.

MPINSIDE_PRINT_SIZ_IN_K
       Print transfer sizes in Kbytes instead of Mbytes  (the default) in the mpinside_stats file.

MPINSIDE_SHOW_READ_WRITE
```

Include in the mpinside_stats file two columns indicating the time, number of char, and number of calls to the libc read(), write() functions. Note this time is already excluded from the "comput" column. Anyway this is also "comput" time, i.e., time spent outside MPI.

MPINSIDE_INIT_CAL MPINSIDE_SYNCHRO_CLOCK

The purpose of these environment variables is to deal with cluster non synchronized clocks. They are only meaningful if MPINSIDE_CLOCK_IS_SYNCHRO is not set or if MPINSIDE_PATTERN_MATCH is not set to TOKEN

MPINSIDE_INIT_CAL <integer values>

If set MPInside will sleep for this time value in seconds just after the MPI_Init() call to be able to calibrate clocks. See also MPINSIDE_SYNCHRO_CLOCK for clock calibration

MPINSIDE_SYNCHRO_CLOCK <Collective MPI function>:heartbeat:method

By default th e clock translation tables are just built when experimentation starts. On Altix or future ICE system the clocks are/will be hardware synchronized. This is not t he case on current clusters. So some translation tables have to be built (see MPINSIDE_SYNCHRO_RETRIES). Unfortunately the clocks may deviate for few micro seco nds with time. To have a 1000 microsecond deviation after 1000 seconds of elapsed time is not uncommon. This deviation is not important when using only intra-CP U timing. This is dramatic when adding this amount of error thousand of times as MPInside does to evaluate the "Send Late time". By setting this variable, the timing translation tables are reinitialized every heartbeat count to the specified Collective MPI function with the right communicator (see MPINSIDE_COMM_ T_W).Method :

a : Synchronize with no correction every heartbeat after the start of the experimentation .

c: Synchronize after the first heartbeat following the start of the experimentation (see MPINSIDE_COLLECTIVE_WINDWOW) and use the correction elements built here for the rest of the experimented run.

A: Synchronize every heartbeat following the start of the experimentation

```
and use  the  new  correction  elements
built here up to the next heartbeat.

i :  Synchronize after the first heartbeat following the start of the
program (not the start of the experimenta-
tion) and use the correction elements built
here for the rest of the experimented run

d : Work with the default (no synchronization, no correction)
but just print a message on stdout every  heartbeat
regardless of the "method"

In  addition  to  any  MPI collective function one can set "MPINSIDE_Collective_call".
In such case the heartbeat
will be based on the number of call the MPINSIDE_Collective_call()
the user inserted in his source code. Note the
function  will do nothing but incrementing a counter
but is has to be fully collective. Note the application must
be linked with this libmpinside_stub.so library in case of a call to this routine.

Examples:  setenv  MPINSIDE_SYNCHRO_CLOCK  MPI_Allgather:100:c;
           setenv  MPINSIDE_SYNCHRO_CLOCK  MPINSIDE_Collective_call:50:a

MPINSIDE_SYNCHRO_RETRIES  <RETRIES>:<TARGET_ERROR(Âµs)>
       MPInside has to be able to translate  any rank clock to another rank<92>s clock.
Establishing a one-to-one
function with the right communicator (see  MPINSIDE_COMM_ T_W).Method :

a : Synchronize with no correction every heartbeat after the start of the experimentation .

c:  Synchronize  after  the  first  heartbeat  following
the  start of the experimentation (see MPINSIDE_COLLEC-
TIVE_WINDWOW) and use the correction elements built here for the rest of the experimented run.

A: Synchronize every heartbeat following the start
of the experimentation and use  the  new  correction  elements
built here up to the next heartbeat.

i :  Synchronize after the first heartbeat following
the start of the  program (not the start of the experimenta-
tion) and use the correction elements built here for the rest of the experimented run
```

```
d : Work with the default (no synchronization, no correction)
but just print a message on stdout every  heartbeat
regardless of the "method"

       In  addition  to  any  MPI collective function one can set "MPINSIDE_Collective_call".
       In such case the heartbeat
       will be based on the number of call the MPINSIDE_Collective_call() the user inserted
       in his source code. Note the function  will do nothing but incrementing a counter
       but is has to be fully collective. Note the application must
       be linked with this libmpinside_stub.so library in case of a call to this routine.

       Examples:  setenv  MPINSIDE_SYNCHRO_CLOCK  MPI_Allgather:100:c;
       setenv  MPINSIDE_SYNCHRO_CLOCK  MPINSIDE_Collective_call:50:a

MPINSIDE_SYNCHRO_RETRIES  <RETRIES>:<TARGET_ERROR(Âµs)>
       MPInside has to be able to translate  any rank clock
       to another rank<92>s clock. Establishing a one-to-one
       common point is necessary for that. This common point can be  more
       or  less  fuzzy.  MPInside  will  make
       RETRIES  attempts  to get an error less than TARGET-ERROR.
       Then the target error will be increased by 5 Âµs
       for RETRIES attempts and so on <85>. Take care not to set
       a too low value as this  will result in  several
       attempts. Example  16:10.0. Default is 8:20.
```

```
Modeling
       MPINSIDE_MODEL PERFECT+<CPU_BOOTS>
              If  set  MPInside,  instead  of  measuring  communications,
              will model them as if the communication engine
              (hardware+MPI) was perfect: Zero latency, infinite bandwidth.
              For example  MPINSIDE  PERFECT+1.20  is  the
              value  we used to get the Paratec perfect interconnect time
              on an Hapertown system with a run on a Clovertown system.

Notes
       MPInside uses library preloading to initialize performance measurement
       and therefore can only be used  with  exe-
       cutables that have been linked dynamically.
```

See also
    libFFIO.so(3)

    The following file and command are listed relative to the MPInside installation path.

    doc/mpinside_3.3_ref_manual.pdf

    bin/pram -h bin/mpinside2wrl -h : Utilities to extract
    information from Binary transfer matrices

    Utility to extract information from Binary transfer matrices

AUTHOR
    Daniel Thomas

COPYRIGHT

MPInside Tool   August 2010   MPInside(3)

Index

C

communication stiffness, 38

E

environment variables, 8
  miscellaneous, 16
environmental variables and stack of features, 8

I

introduction, 1

L

loading the MPInside module, 2

M

miscellaneous environment variables, 16
MPI communication terminology
  branch, 4
  branch partner, 4
  function time (FT), 3
  function waiting time (FWT), 3
  ordinary branches, 4
  recv branches, 5
  send branches, 4
  send late time (SLT), 4
  transfer time (Tt), 3
  wait branches, 4
MPInside Binary Transfer Matrices, 29
MPInside(3) man page, 49
mpinside_clstk_post.xxx files, 30
mpinside_stats file, 21
  bytes transferred and number of requests arrays, 24
  columns meaning, 23
  user counters, 22

N

non-synchronized send/receive pair definition and terminology, 3

O

obtain an MPInside report
  see how often collective functions are called, 47
overview, 1

P

partner branches
  ordinary, 35
  recv, 36
  send, 35
  wait, 36
perfect interconnect zero latency infinite bandwidth, 40
pinside_stats_.M-N files, 26
post-processing, 6
pram utility, 30

R

restraining the profile to selected parts of the application, 7

S

see how often collective functions are called, 47
select a window of observation
  re-compilation, 45
  spy the collective functions, 46

U

using the MPInside tool
  environmental variables and stack of features, 8
  post-processing, 6
  SGI MPT, 5
  X86 HP–MPI, 6
  X86 Intel MPI, 6
  X86 SCALI MPI, 6

V

viewing the MPInside(3) man page, 2