Log File Formats for Parallel Applications: A Review

Article *in* International Journal of Parallel Programming · April 2009

DOI: 10.1007/s10766-009-0093-x · Source: DBLP

Athanasios I Margaris
University of Thessaly
55 PUBLICATIONS **172** CITATIONS

Int J Parallel Prog (2009) 37:195–222
DOI 10.1007/s10766-009-0093-x

Log File Formats for Parallel Applications: A Review

Athanasios I. Margaris

Received: 9 September 2008 / Accepted: 8 January 2009 / Published online: 17 February 2009

Abstract The objective of this paper is the review of the log file formats that allow the performance visualization of parallel applications based on the usage of message passing interface (MPI) standard. These file formats have been designed by the LANS (Laboratory for Advanced Numerical Software) group of the Argonne National Laboratory and they are distributed together with the corresponding viewers as part of the MPE (multipurpose environment) library of the MPICH implementation of the MPI. The formats studied in this paper is the ALOG, CLOG, SLOG1 and SLOG2 file formats—the formats are studied in chronological order and the main features of their structures are presented.

Keywords Parallel programming · Message passing interface · Log files

1 Introduction

One of the most important tasks associated with the execution of a parallel application is the collection of the process events and their storage to special file types known as log files. In most cases these files keep for each event its type and its time stamp, i.e. the time instance of its occurrence. These files are created separately by the processes of the parallel application, and after the program termination they are merged to a single log file for further processing and viewing.

There are many applications that have been designed to generate such file types and process them to estimate the performance of parallel applications. Typical examples of such applications are the Intel Trace Analyzer [1] and the UTE (Unified Tracing

A. I. Margaris (✉)
Department of Applied Informatics, University of Macedonia, 156 Egnatia St.,
54006 Thessaloniki, Greece
e-mail: amarg@uom.gr

Environment) library of the IBM [10] that can create log files by logging the appropriate event types. On the other hand, the visualization of those files, namely, the display of their contents in the computer screen, can be performed by a lot of applications, such as the ParaGraph [7], the VIZIR [6], the VISPAT [9] and the TraceView [12]. The Argonne National Laboratory has developed also, a set of viewers for the ALOG, CLOG, SLOG1 and SLOG2 file formats studied here, identified by the names upshot [8], nupshot [11], and Jumpshot [15] (Jumpshot comes in three variances, namely, Jumpshot-2, Jumpshot-3 and Jumpshot-4, that allow the views of the files CLOG, SLOG1 and SLOG2, respectively). The detailed description of those viewers are out of the scope of this paper.

Generally speaking, there are three types of logging procedures: (a) event-based logging which is the simplest logging operation: for each event, its name as well as the corresponding time stamp are stored to the log file (b) state-based logging: in this logging type the start and the end time of each event—that uniquely define a state—are stored in the log file (c) drawable-based logging: in this logging type a whole record is stored for each event that contains information not only about the event itself but also about the way it will be drawn in the computer screen.

The drawing of the log files in the screen displays their contents in the form of a special graph type known as space time diagram. This is a 2D diagram in which the horizontal axis represents the time axis with the zero point to indicate the start time of the application, while the vertical axis can be organized in many different ways that give rise to the various view types used for the preview of the log data, such as the thread-activity view, the processor-activity view, the thread-processor view and the processor-thread view. In the most general case, the space time diagram shows a horizontal line per process displaying marks for the start and the end time for each event (for the event-logging type) of filled rectangles for the duration of each state of that process (for the state-logging type). The last graph type (shown in Fig. 1) is very similar to the Gantt diagrams and its construction requires the conversion of the trace files created by the logging mechanism to interval files. This file type contains a set of interval records each one of them is characterized as 'begin interval', 'end interval', 'continuation interval' and 'complete interval' [14]. The complete interval type is the most simple type and it is associated with MPI calls that are executed without interruptions by a simple processor. On the other hand the continuation and the end intervals are related to MPI calls that are characterized by pauses during execution and therefore they are associated with many different intervals. In this case all the intermediate records are marked as continuation records while the last record for these MPI calls is marked as the end—i.e. the last—record.

In Fig. 2 the structure of a typical file logging generation mechanism is shown. From this figure it is clear that the generation of the log files is based on the usage of a trace library that has to be compiled and linked to the executable file of the parallel application. After the generation of the log files—one file per process—they are merged to create the resulting file whose data are then processed to extract statistical information about the operation of the parallel application. In most cases this file is converted to the SLOG file format [14]—the most commonly used log file type—that will be studied in details in the following sections.

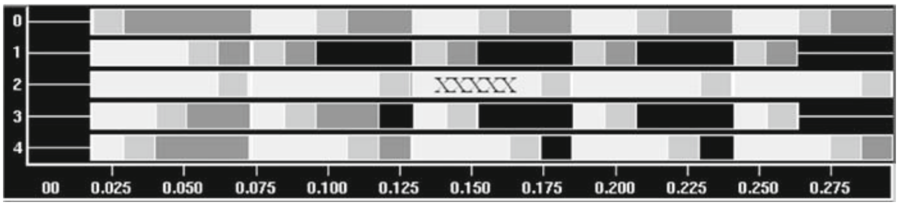


Fig. 1 The structure of a typical space time diagram. The XXXXX region represents a state associated with the process rank p = 2 with start time $T_1 = 0.125$ and end time $T_1 = 0.175$

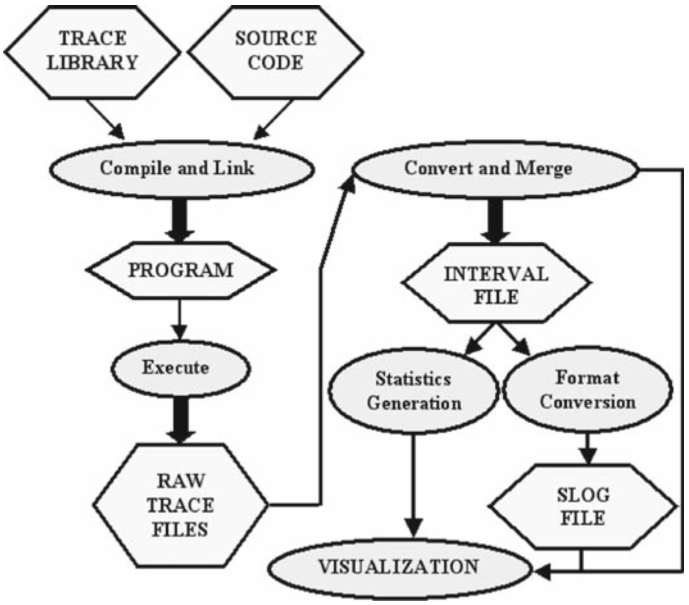


Fig. 2 The structure of the log file generation mechanism

2 The ALOG File Format

The ALOG format is the oldest and simplest log file format and it is not used in our days—therefore its description is included only for the sake of completeness. This ASCII file contains for each recorded event a single line of parameter values (usually there is no need to use all the values) separated by null characters [8]. The type and the meaning of those parameters are presented bellow:

- Event Type: describes a user-defined event type
- Process Id: identifies the process associated with the current event
- Task Id: identifies the task associated with the current event
- Integer Data: user-defined data related to the current event
- Clock Cycle: allows the discrimination of events occurred at different time cycles

– Time Stamp: identifies, in conjunction with the previous field, the event time stamp
– String Data: represents user-defined string data

Before the storage of those lines, a set of other lines associated with the so-called pseudo-events are written in the file. These pseudo-events are described by negative integers and they are associated with important system parameters. The most important pseudo-events, their parameters, and the meaning carried by them, are presented below:

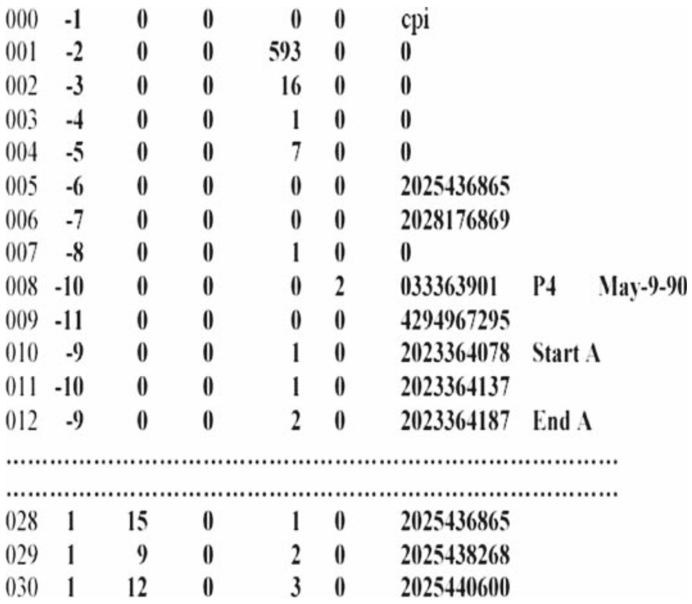
– Pseudo-event [-1]: the 'string data' field identifies the program that created the log file as well as the creation date.
– Pseudo-event [-2]: the 'integer data' field contains the number of events that have been recorded to the log file.
– Pseudo-event [-3]: the 'integer data' field contains the number of processes whose events have been recorded.
– Pseudo-event [-4]: the 'integer data' field contains the number of tasks of the parallel application.
– Pseudo-event [-5]: the 'integer data' field contains the number of different event types.
– Pseudo-event [-6]: the 'time stamp' field contains the time instance of the first recorded event.
– Pseudo-event [-7]: the 'time stamp' field contains the time instance of the last recorded event.
– Pseudo-event [-8]: the 'integer data' field contains the number of the completed clock cycles.
– Pseudo-event [-9]: the 'integer data' field contains the type of the current event while the 'string data' field contains the description of the current event.
– Pseudo-event [-10]: the 'integer data' field contains the type of the current event while the 'string data' field contains a string that can be printed via the 'printf' function.
– Pseudo-event [-11]: the 'time stamp' field contains the time instance for which the system timer has been reset.

The contents of a typical ALOG file are shown in Fig. 3. The interpretation of the displayed lines is based on the previously given description. For example, line 001 says that the log file contains 593 logged data, while according to the line 008, the log file was created by the p4 application in 9-May-1990. The interpretation of the remaining lines is done in the same way.

The preview of the ALOG files in a graphical way can be done by using the upshot and nupshot X Windows applications.

3 The CLOG File Format

Even through the ALOG files allow the fast access of their data in the main memory, the fixed structure of their records makes the addition and usage of new fields, a very difficult task. To overcome this limitation, the CLOG file format has been created; this



```
000  -1   0   0     0  0   cpi
001  -2   0   0   593  0   0
002  -3   0   0    16  0   0
003  -4   0   0     1  0   0
004  -5   0   0     7  0   0
005  -6   0   0     0  0   2025436865
006  -7   0   0     0  0   2028176869
007  -8   0   0     1  0   0
008 -10   0   0     0  2   033363901   P4    May-9-90
009 -11   0   0     0  0   4294967295
010  -9   0   0     1  0   2023364078  Start A
011 -10   0   0     1  0   2023364137
012  -9   0   0     2  0   2023364187  End A
........................................................................
........................................................................
028   1  15   0     1  0   2025436865
029   1   9   0     2  0   2025438268
030   1  12   0     3  0   2025440600
```

Fig. 3 The contents of a typical ALOG file

format allows the generation of more sophisticated log files by applying the following rules:

(1) The logging of the application data is based on the usage of the MPI_Wtime MPI function [13] that returns the elapsed time between two time instances.
(2) For performance reasons, the logged data are not stored in hard disk files, but in appropriate buffers in the main memory. If these buffers are full of data and they are not able to store more of them, additional memory is allocated through the ‘malloc’ function. However, for very large log files the periodic memory dump to the hard disk is necessary.
(3) When the logging procedure completes, each process adds to its own records additional information—such as its PID—and then all these records are adjusted in order to be synchronized each other.
(4) The resulting CLOG file is generated by merging the log files created by the system processes to a single file. This merging requires the virtual placement of the system processes to the nodes of a binary tree [4] with the process with rank $R = 0$ to be placed in the root node (if the number of processes is not a power of 2 a value of -1 is assigned to the unused nodes). Then, the processes of the leaf nodes send their data to their parent process—by calling the MPI data transfer functions—which, in turn, merges the memory blocks received from its two children and send them to its own parent process located in the next level. In the last step, the root process receives the two halves of the process data, merges them to a single buffer and stores them to the log file by using the external32 MPI data representation [13]. The creation of the CLOG file is shown in a graphical way in Fig. 4.

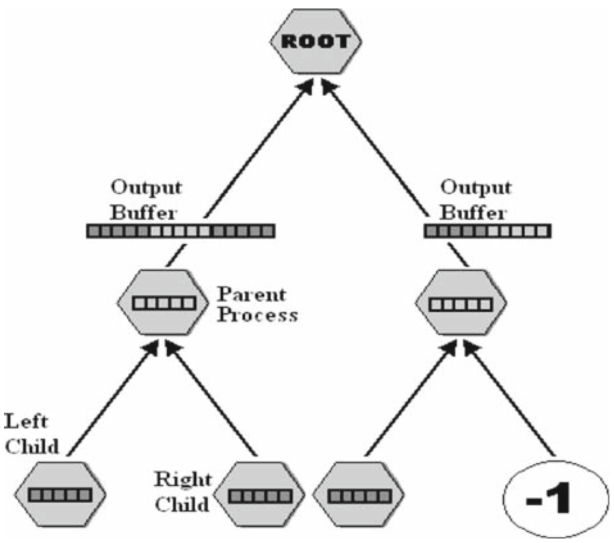
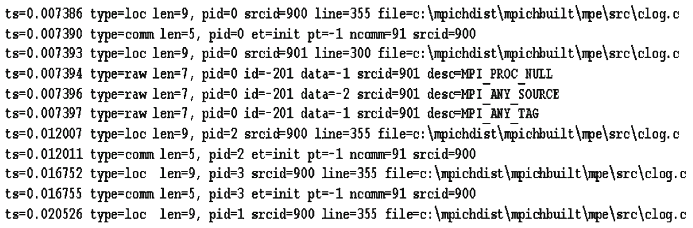


Fig. 4 The generation of a CLOG file by a group of processes

The CLOG data organization is based on the single linked list data structure with each list node—in the current MPICH implementation—to be capable of storing up to 65536 bytes of data. Regarding the structure of those nodes, they can store a set of different records that share a common header. This header is composed of a set of fields the most important of them are the event time stamp, the record type, and the rank of the process associated with the current event in the communicator MPI_COMM_WORLD. The record types supported by the CLOG file format, are the following:

- CLOG_ENDLOG: identifies the end of the log file
- CLOG_ENDBLOCK: identifies the end of a list node
- CLOG_UNDEF: unknown/undefined event
- CLOG_RAWEVENT: a general purpose event record
- CLOG_MSGEVENT: an event associated with message passing operations
- CLOG_COLLEVENT: an event associated with collective operations
- CLOG_COMMEVENT: an event associated with communicator operations
- CLOG_EVENTDEF: allows the description of a system event
- CLOG_STATEDEF: allows the description of a system state
- CLOG_SRCLOC: allows the identification of a source code line
- CLOG_SHIFT: used in time shift calculations

After the above common header, the actual records follow, that contain the data of the log file. There are eight different record types—each one with its own fields—corresponding to the most of the record types described above. The definition of those records in C programming language can be found in the MPICH source code and it is not reproduced here for sake of brevity. It is simply pointed out that the fields



```
ts=0.007386 type=loc len=9, pid=0 srcid=900 line=355 file=c:\mpichdist\mpichbuilt\mpe\src\clog.c
ts=0.007390 type=comm len=5, pid=0 et=init pt=-1 ncomm=91 srcid=900
ts=0.007393 type=loc len=9, pid=0 srcid=901 line=300 file=c:\mpichdist\mpichbuilt\mpe\src\clog.c
ts=0.007394 type=raw len=7, pid=0 id=-201 data=-1 srcid=901 desc=MPI_PROC_NULL
ts=0.007396 type=raw len=7, pid=0 id=-201 data=-2 srcid=901 desc=MPI_ANY_SOURCE
ts=0.007397 type=raw len=7, pid=0 id=-201 data=-1 srcid=901 desc=MPI_ANY_TAG
ts=0.012007 type=loc len=9, pid=2 srcid=900 line=355 file=c:\mpichdist\mpichbuilt\mpe\src\clog.c
ts=0.012011 type=comm len=5, pid=2 et=init pt=-1 ncomm=91 srcid=900
ts=0.016752 type=loc  len=9, pid=3 srcid=900 line=355 file=c:\mpichdist\mpichbuilt\mpe\src\clog.c
ts=0.016755 type=comm len=5, pid=3 et=init pt=-1 ncomm=91 srcid=900
ts=0.020526 type=loc  len=9, pid=1 srcid=900 line=355 file=c:\mpichdist\mpichbuilt\mpe\src\clog.c
```

Fig. 5 The contents of a CLOG file as they are printed by the clog_print utility

of these records are associated with various system parameters such as the type and the length of the records, the process id, the communicator, and the associated event type.

CLOG files are binary files and they cannot be viewed by ASCII editors. The contents of those files can be viewed by using the Jumpshot-2 X Windows application [15]. However, the MPICH implementation of the MPI standard includes a CLOG viewer named clog_print that gets as command line argument the name of a CLOG file and prints in text mode the file contents. A typical example of the clog_print command is shown in Fig. 5.

4 Scalable File Formats

The main drawback of the file formats presented in the previous sections, is the serial search of the data to be plotted: if a specific file portion is to be drawn in the computer screen, the program must read all the previous data sections, a procedure that causes very long time delays for very large log files. To overcome this limitation a new file format has been designed that allows the file access in a random way: this means that the program reads directly the data set to be plotted without needing to traverse the previous values. An additional feature of this new file format is the independence of the time duration of the data retrieval operation from the file size and from the position of the retrieved data inside the file. This new file format is known as scalable log file format, or in short, as SLOG format.

There are two different scalable log file formats, from which the SLOG1 format [14] is used in state-based logging while the SLOG2 format [3] is used in drawable-based logging. In SLOG1 file format, the logged data are not stored in a serial fashion. More specifically, the time duration of the logging procedure is separated into small time intervals—known as time frames—that are then stored in contiguous memory locations and contain references to the memory blocks containing the actual data. When the user wants to plot to the screen a specific file portion, the program identifies very quickly the associated time frames and then by following their references, reads the logged data values to be plotted. If the log file is very large, the time frames are grouped into frame directories that are used in the same way. On the other hand, in SLOG2 file format, the records and the associated time intervals are stored to the nodes of a binary tree.

5 The SLOG1 File Format

The structure of the SLOG1 file type is composed of a file header, three tables (identified by the names ‘display profile table’, ‘thread table’, and ‘interval record definition table’), and two sections (namely the ‘statistics section’ and the ‘data section’) that are used in the following way:

The file header contains general purpose information such as the version of the API used for file creation, the maximum number of time frames the file can keep, and a set of pointers to the various file sections that allow their access by the file viewer.

The display profile table contains for each different record type a line composed of information about the plotting of this record type, such as the drawing color and the legend title.

The thread table contains information about the threads of the parallel application.

The interval record definition table contains for each different record type a set of parameter values such as the start time and the record time duration as well as its type and the category it belongs.

The statistics section contains information associated with the distribution of each record type to a set of time bins, allowing thus the generation of histograms and the estimation of statistical parameters.

The data section contains the actual data associated with each time frame of the SLOG1 file.

The structure and the properties of all these file sections are described in details in the next pages.

5.1 The SLOG1 File Header

The SLOG1 file header is a C structure that contains general purpose file attributes. These attributes in short, are the following:

1. A file pointer that is used by the SLOG API library.
2. A structure of four pointers from which the first, the second and the third pointer identify the first, the current, and the last byte of the header, respectively, while, the fourth pointer identifies the SLOG file itself.
3. A 2-integer data structure containing the major and the minor version number of the SLOG API.
4. An integer field identifying the frame size in bytes (the default value of this field is the value 256).
5. An integer field containing the size (in bytes) of the memory buffers that keep the pseudo-record data values. These records are only used if the file records are stored in ascending order with respect to the ‘start time’ field.
6. An integer variable that keeps the maximum number of frames that can be stored in a frame directory.
7. A boolean variable that has a value of TRUE if the file records are stored in ascending order with respect to the ‘start time’ field, or a value of FALSE, otherwise.
8. A boolean variable that has a value of TRUE if the file records are stored in ascending order with respect to the ‘end time’ field, or a value of FALSE, otherwise.

9. A boolean variable that has a value of TRUE if the memory buffers reserved for the pseudo-records have been used during the SLOG1 file creation, or FALSE, otherwise.
10. Six pointers that identify the offsets inside the file of the first byte for the various file sections.

5.2 The Interval Record Definition Table

For each one of the different record types associated with a SLOG1 file, the interval record definition table contains the attributes of those records such as their types and the value of the begin-end bits (bebits). This table is described by an appropriate structure that keep the following information:

- the maximum number of record types in the table
- the number of records currently available
- the amount of the additional memory allocated by the program each time the current number of records has reached its maximum number, and there is no free space to keep new records; therefore the structure is not characterized by space limitations.
- the offset in bytes with respect to the beginning of the file, of the position of a newly inserted record.
- The data of the different record types that are stored in an array of structures, each one of them keeps the following information:
    - the record type
    - the record interval type. Possible values are the following:
        - (0,0) for a 'continuation' interval
        - (0,1) for an 'end' interval
        - (1,0) for a 'begin' interval
        - (1,1) for a 'complete' interval
    - the number of the associated records (for continuation intervals)
    - the number of the arguments of the associated MPI call

The contents of a typical interval record definition table as they are printed by the slog_print MPICH utility are shown in Fig. 6.

5.3 The Display Profile Table

The display profile table has the same structure as the interval record definition table and keeps for each record type a set of parameter values that allow the display of the records of that type in an X-Window screen using the Jumpshot-3 application. The only difference between this table and the previous one concerns the information kept for each record type. Besides the 'record type' and the 'record interval type' which are the same as the previous case the remaining items are as follows:

- the drawing color of the current record type
- the legend text associated with the current record type

```
The Interval Record Definition Table   The Display Profile

  def[00] = [ 100, (1, 1), 0, 0 ]      def[00]=[100,(1,1),state,label1,green]
  def[01] = [ 101, (1, 1), 0, 1 ]      def[01]=[101,(1,1),state,label2,blue:("arg_0")]
  def[02] = [ 102, (1, 1), 0, 2 ]      def[02]=[102,(1,1),state,label3,yellow:("arg_0" "arg_1") ]
  def[03] = [ 201, (1, 1), 0, 1 ]      def[03]=[201,(1,1),state,label5,gold:("arg_0")]
  def[04] = [ 202, (1, 1), 0, 2 ]      def[04]=[202,(1,1),state,label6,RoyalBlue:("arg_0" "arg_1")]
  def[05] = [ 300, (1, 1), 0, 0 ]      def[05]=[300,(1,1),state,label7,navy]
  def[06] = [ 301, (1, 1), 0, 1 ]      def[06]=[301,(1,1),state,label8,pink:("arg_0")]
  def[07] = [ 302, (1, 1), 0, 2 ]      def[07]=[302,(1,1),state,label9,grey:("arg_0" "arg_1")]
  def[08] = [ 401, (1, 1), 0, 1 ]      def[08]=[401,(1,1),state,label11,LimeGreen:("arg_0")]
  def[09] = [ 402, (1, 1), 0, 2 ]      def[09]=[402,(1,1),state,label12,brown:("arg_0" "arg_1")]
  def[10] = [ 500, (1, 1), 0, 0 ]      def[10]=[500,(1,1),state,label13,NavyBlue]
  def[11] = [ 501, (1, 1), 0, 1 ]      def[11]=[501,(1,1),state,label14,ivory:("arg_0")]
  def[12] = [ 502, (1, 1), 0, 2 ]      def[12]=[502,(1,1),state,label15,AliceBlue:("arg_0" "arg_1")]
  def[13] = [ 600, (1, 1), 0, 0 ]      def[13]=[600,(1,1),state,label160,MistyRose]
  def[14] = [ 700, (1, 1), 0, 0 ]      def[14]=[700,(1,1),state,label170,linen]
  def[15] = [ 998, (1, 1), 0, 0 ]      def[15]=[998,(1,1),Message,Forward Arrow, white]
  def[16] = [ 999, (1, 1), 0, 0 ]      def[16]=[999,(1,1),Message,Backward Arrow, grey]
```

Fig. 6 Typical contents of the record definition table and the display profile table of an SLOG1 file

– the class of the current record type. There are three different classes the most important of them are the following:

  - MPI state: the current record is associated with collective operations, such as barrier synchronization, broadcasting, scattering, gathering and reduction. The collective operations are drawn as horizontal lines whose ends identify the start time and the end time of the associated operation
  - Messase: the record is associated with point to point operations between process pairs. These point to point operations are plotted as directed arrows that connect the time lines associated with the sending and the receiving processes; regarding the start time and the end time of the point to point operation they are identified by the x-coordinate of the arrow ends.

The contents of a typical display profile table as they are printed by the slog_print MPICH utility are shown in Fig. 6, too.

5.4 The Thread Table

The last table of the SLOG1 file format, is the thread table that keeps information about the threads of the parallel application. This table is described by a structure that has the same fields with the previous ones; regarding the thread information it is stored to the cells of an array of structures with the following fields:

– an identifier that describes the process in the MPI environment
– an identifier that describes the thread in the MPI environment
– an identifier that describes the process in the operating system environment
– an identifier that describes the thread in the operating system environment
– an identifier that describes the parallel application itself

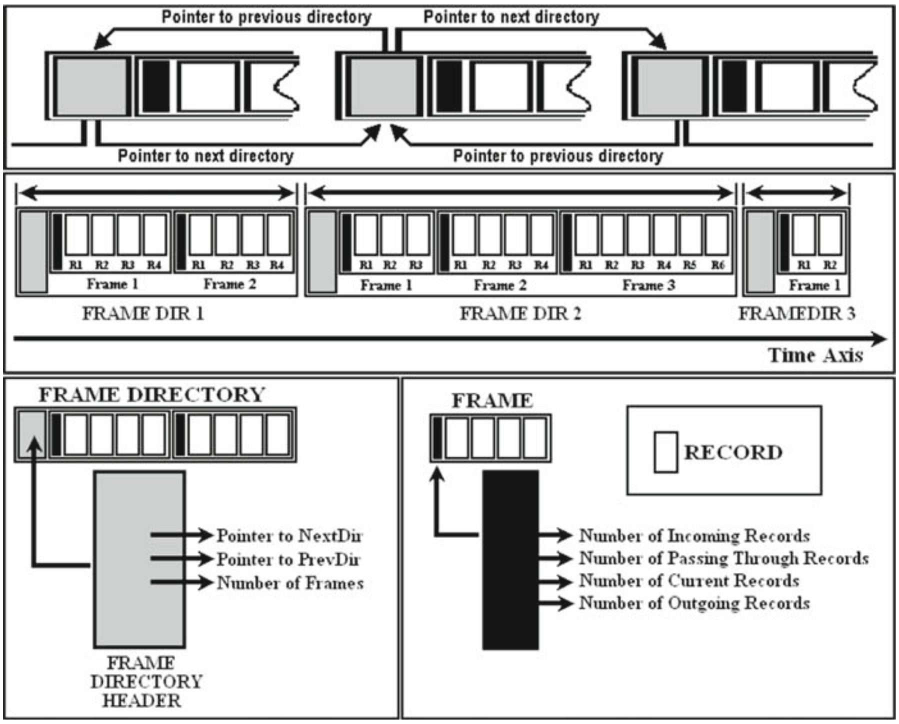

Pointer to previous directory
Pointer to next directory
Pointer to next directory
Pointer to previous directory
R1 R2 R3 R4
Frame 1
Frame 2
FRAME DIR 1
FRAME DIR 2
FRAMEDIR 3
Frame 3
R5 R6
Time Axis
FRAME DIRECTORY
Pointer to NextDir
Pointer to PrevDir
Number of Frames
FRAME DIRECTORY HEADER
FRAME
RECORD
Number of Incoming Records
Number of Passing Through Records
Number of Current Records
Number of Outgoing Records

Fig. 7 The organization of the data section of the SLOG1 file format

The contents of the thread table can be viewed by the slog_print MPICH utility, and they are similar with the ones associated with the record definition and the display profile tables.

5.5 The Data Section

The organization of the data section of the SLOG1 file is characterized by the creation of a set of frame directories that are connected together in a double linked list fashion; their headers contain pointers showing the previous and the next directory, as well as an additional field that keeps the number of frames in each frame directory, that is not empty. Regarding the data of each frame they are stored to the cells of an array of records of the appropriate type. Each one of those records keeps the start and the end time for each frame as well as the offset of the frame data inside the frame directory it belongs. The organization of the SLOG1 data file is shown in Fig. 7.

According to the relationship between the start and the end time of a data frame, T_s^f and T_e^f and the corresponding parameters of a frame record, T_s^r and T_e^r, we can distinguish between four different record types: (a) current records that satisfy the property $T_s^r > T_s^f$ and $T_e^r < T_e^f$ (b) incoming records that satisfy the property $T_s^r < T_s^f$ and $T_e^r < T_e^f$ (c) passing through records that satisfy the property $T_s^r < T_s^f$ and $T_e^r > T_e^f$

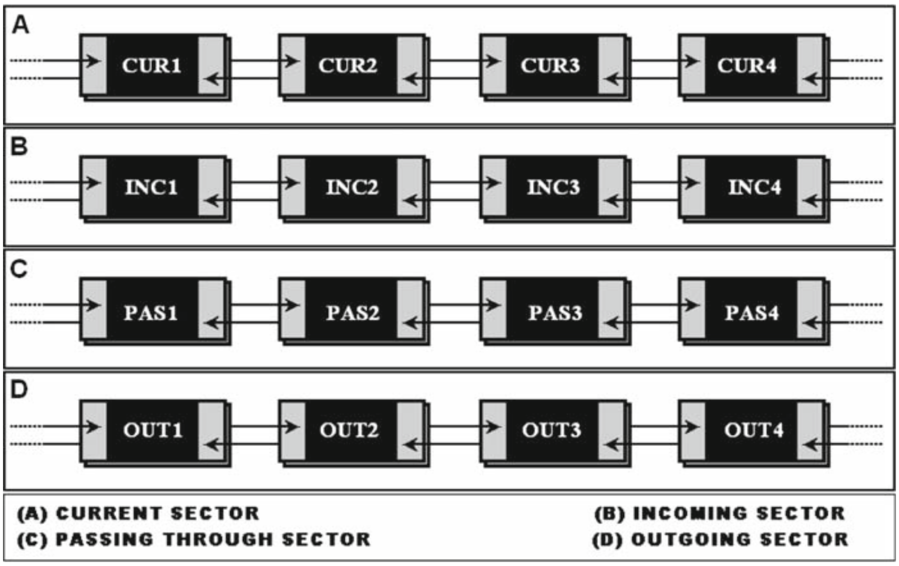


Fig. 8 The four sectors associated with each data frame of the SLOG1 file

and (d) outgoing records that satisfy the property $T_s^r > T_s^f$ and $T_e^r > T_e^f$. Therefore, each frame maintains four double linked lists—the current sector, the incoming sector, the passing through sector and the outgoing sector—that keep the frame records of each one of the above record types. This organization concerns the data section of each frame and it is shown in Fig. 8. A frame header is also available that keeps for each record type the number of records of that type and the size in bytes of the corresponding sector. Figure 9 shows the structure of the data section of the whole SLOG file in a more detailed level of description.

The last structure of interest that will be presented here, is the one that is used to describe each one of the frame records; this structure is named SLOG_intvlrec_r and contains the following information:

- the record size in bytes. If this field has a value of zero the current record is characterized as a null record.
- the class of the current record.
- the type of the current record; the value of this field must belong to the record definition table.
- the interval type of the current record that allows to characterize a record as a begin, end, complete or continuation interval.
- the start time T_{rec} of the current record
- the duration D_{rec} of the current record. If this field is added to the 'start time' field, the end time of the record can be calculated allowing thus the insertion of the record to the current, incoming, passing through or outgoing sector.
- the node id, the cpu id and the thread id of the source process in point to point communications.

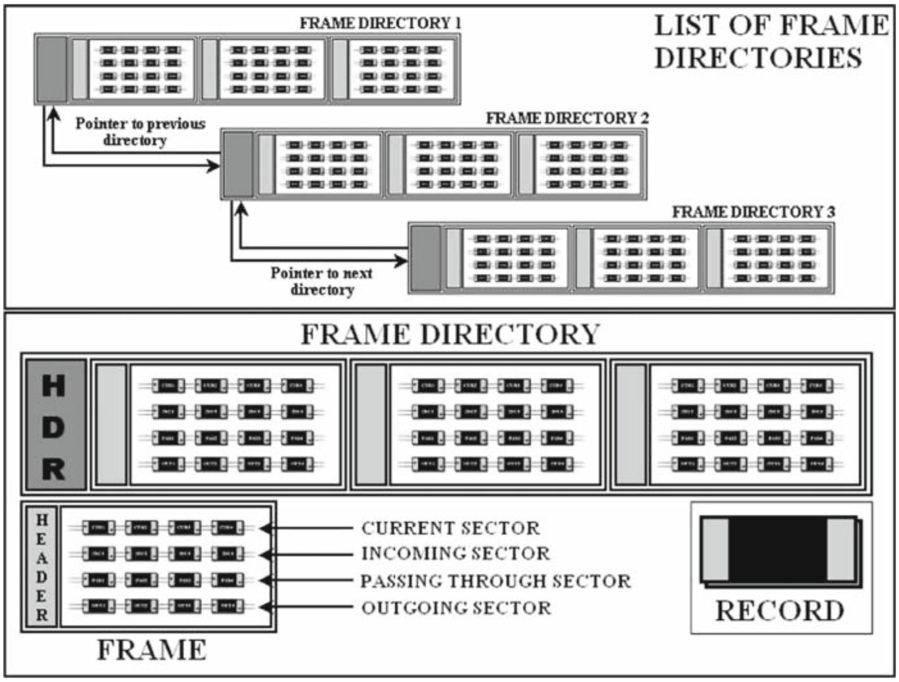


Fig. 9 The organization of the data section of the SLOG1 file—A detailed view

– the node id, the cpu id and the thread id of the target process in point to point communications.
– the instruction address associated with the current record.
– the number of records associated with the current record.
– a set of pointers to the associated records.

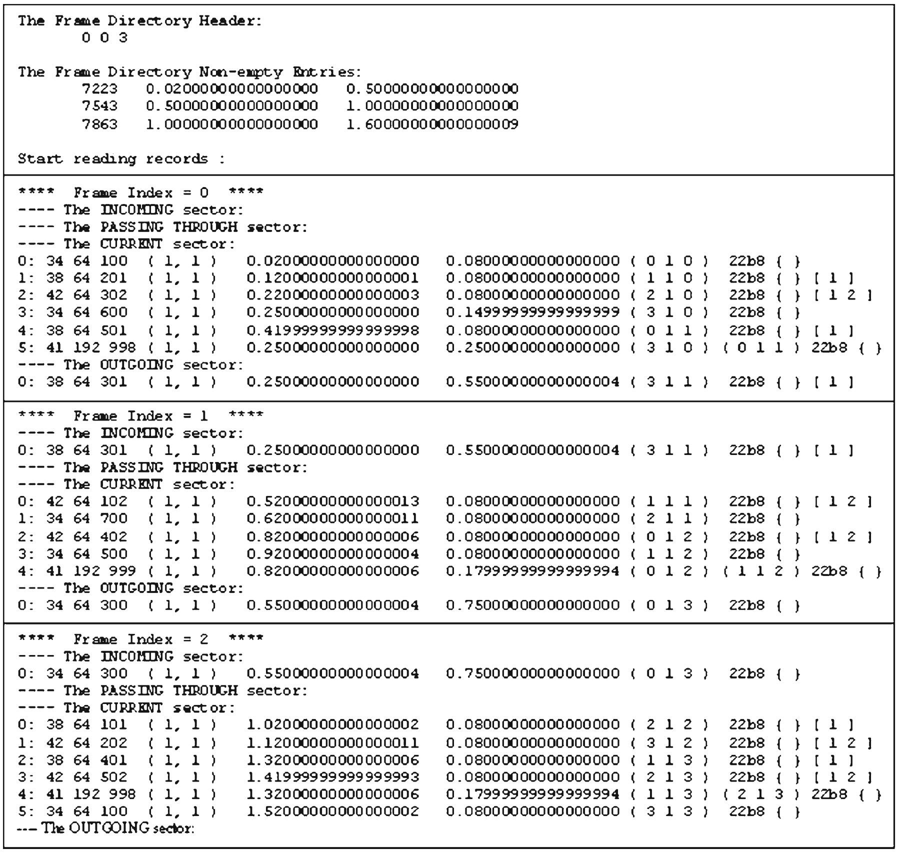
The contents of a typical SLOG1 data section as they are printed by the slog_print MPICH utility are shown in Fig. 10.

5.6 The Statistics Section

The statistics section of the SLOG1 file allows the generation of histograms that show the time distribution of the different record types. To calculate this distribution, the time axis is divided into a set of time bins, and each record type is associated with one or more bins according to its time duration. If a time bin is totally overlapping with the interval that gives the duration of the record, a value of 1 (corresponding to 100%) is assigned to that bin. On the other hand, for partial overlapping a decimal number in the interval (0,1) that expresses the percentage of the overlapping, is used.

The parameters associated with the statistics section are stored in a data structure that keeps the following information:

– the start time T_{start} of the statistics operation; this value is equal to the value of the 'start time' field of the first frame associated with the first frame directory.



```
The Frame Directory Header:
       0 0 3

The Frame Directory Non-empty Entries:
       7223   0.02000000000000000   0.50000000000000000
       7543   0.50000000000000000   1.00000000000000000
       7863   1.00000000000000000   1.60000000000000009

Start reading records :

****  Frame Index = 0  ****
---- The INCOMING sector:
---- The PASSING THROUGH sector:
---- The CURRENT sector:
0: 34 64 100  ( 1, 1 )   0.02000000000000000   0.08000000000000000 ( 0 1 0 )  22b8 { }
1: 38 64 201  ( 1, 1 )   0.12000000000000001   0.08000000000000000 ( 1 1 0 )  22b8 { } [ 1 ]
2: 42 64 302  ( 1, 1 )   0.22000000000000003   0.08000000000000000 ( 2 1 0 )  22b8 { } [ 1 2 ]
3: 34 64 600  ( 1, 1 )   0.25000000000000000   0.14999999999999999 ( 3 1 0 )  22b8 { }
4: 38 64 501  ( 1, 1 )   0.41999999999999998   0.08000000000000000 ( 0 1 1 )  22b8 { } [ 1 ]
5: 41 192 998 ( 1, 1 )   0.25000000000000000   0.25000000000000000 ( 3 1 0 ) ( 0 1 1 ) 22b8 { }
---- The OUTGOING sector:
0: 38 64 301  ( 1, 1 )   0.25000000000000000   0.55000000000000004 ( 3 1 1 )  22b8 { } [ 1 ]

****  Frame Index = 1  ****
---- The INCOMING sector:
0: 38 64 301  ( 1, 1 )   0.25000000000000000   0.55000000000000004 ( 3 1 1 )  22b8 { } [ 1 ]
---- The PASSING THROUGH sector:
---- The CURRENT sector:
0: 42 64 102  ( 1, 1 )   0.52000000000000013   0.08000000000000000 ( 1 1 1 )  22b8 { } [ 1 2 ]
1: 34 64 700  ( 1, 1 )   0.62000000000000011   0.08000000000000000 ( 2 1 1 )  22b8 { }
2: 42 64 402  ( 1, 1 )   0.82000000000000006   0.08000000000000000 ( 0 1 2 )  22b8 { } [ 1 2 ]
3: 34 64 500  ( 1, 1 )   0.92000000000000004   0.08000000000000000 ( 1 1 2 )  22b8 { }
4: 41 192 999 ( 1, 1 )   0.82000000000000006   0.17999999999999994 ( 0 1 2 ) ( 1 1 2 ) 22b8 { }
---- The OUTGOING sector:
0: 34 64 300  ( 1, 1 )   0.55000000000000004   0.75000000000000000 ( 0 1 3 )  22b8 { }

****  Frame Index = 2  ****
---- The INCOMING sector:
0: 34 64 300  ( 1, 1 )   0.55000000000000004   0.75000000000000000 ( 0 1 3 )  22b8 { }
---- The PASSING THROUGH sector:
---- The CURRENT sector:
0: 38 64 101  ( 1, 1 )   1.02000000000000002   0.08000000000000000 ( 2 1 2 )  22b8 { } [ 1 ]
1: 42 64 202  ( 1, 1 )   1.12000000000000011   0.08000000000000000 ( 3 1 2 )  22b8 { } [ 1 2 ]
2: 38 64 401  ( 1, 1 )   1.32000000000000006   0.08000000000000000 ( 1 1 3 )  22b8 { } [ 1 ]
3: 42 64 502  ( 1, 1 )   1.41999999999999993   0.08000000000000000 ( 2 1 3 )  22b8 { } [ 1 2 ]
4: 41 192 998 ( 1, 1 )   1.32000000000000006   0.17999999999999994 ( 1 1 3 ) ( 2 1 3 ) 22b8 { }
5: 34 64 100  ( 1, 1 )   1.52000000000000002   0.08000000000000000 ( 3 1 3 )  22b8 { }
--- The OUTGOING sector:
```

Fig. 10 The contents of a typical SLOG1 data section as they are printed by the slog_print utility

– the end time T_{end} of the statistics operation; this value is equal to the value of the 'end time' field of the last frame associated with the last frame directory. It is clear, that the difference $T_{stat} = T_{end} - T_{start}$ gives the duration of the statistics operation. For sake of brevity, in the following description it is assumed that the SLOG1 file contents only one frame directory; however the generalization for more frame directories is straightforward.
– the number of statistics M to be calculated. Since this calculation is performed for each different record type, the value of this field must be equal to the number of entries of the interval record definition table.
– the number of time bins N, to which the duration of the statistics T_{stat} is going to be divided into. If this value is not set, a default value of 128 is used. The maximum value of this parameter is is equal to 512.
– the width in time W for each time bin. If the number of time bins N is greater than one, the value of this variable is calculated as

$$W = \frac{T_{stat}}{N} = \frac{T_{end} - T_{start}}{N} \tag{1}$$

while, if there is only one time bin, this parameter is simply estimated as

$$W = T_{stat} = T_{end} - T_{start} \tag{2}$$

– the different record types of the SLOG1 file; these values are stored to the buffer ‘Idx’ of unsigned integers with a length of M.
– the statistics values for each different record type; these values are stored to the buffer ‘Sets’ that contains M records—these records are identified by the name SLOG_statset_t—each one of them keeps the following information:

- the interval record class; this value is equal to the value of the corresponding field of the structure SLOG_intvlrec_r that describes the current record.
- the interval record type; this value is equal to the value of the corresponding field of the interval record definition table as well as the display profile table.
- the text of the legend that is going to be printed to the computer screen for this record type. It has the same value with the corresponding field of the display profile table.
- the number of time bins used by the statistics; its value is the same for all the different record times and equal to the number of bins N defined above.
- the statistics information calculated for each record type and for each time bin. This information is stored in the ‘Bins’ array of floating points with a length of N. The cells of this array have a value of 1 for total overlapping between the corresponding time bin and the duration of the current record, or a decimal value in the interval (0,1) for partial overlapping.

The initialization of the statistics section requires the counting of the different record types that are characterized as ‘complete intervals’ and the determination of their number. In the next step, the required memory for the buffers ‘Ind’ and ‘Sets’ is allocated and the buffers are initialized in the following way:

– for each cell of the ‘Sets’ buffer, the record interval type and the legend text are set to the corresponding values of the display profile table.
– for each cell of the ‘Sets’ buffer the ‘Bins’ array of length N is allocated.
– the complete set of the M different record interval types, are copied from the display profile table to the M cells of the ‘Ind’ buffer.

After this initialization procedure, the following algorithm is applied for each record type:

(1) The interval record type R is read from the display profile table and then the value of the Rth cell of the ‘Ind’ buffer is retrieved according to the assignment $L = \text{Ind}[R]$.
(2) The Lth SLOG_statset_t structure of the ‘Sets’ array is retrieved by an operation in the form $S = \text{Sets}[L]$.
(3) The values of the parameters T'_{start} and T'_{end} are estimated as

$$T'_{start} = T_{rec} - T_{start} \tag{3}$$

$$T'_{end} = T'_{start} + D_{rec} \tag{4}$$

where T_{rec} and D_{rec} are the record start time and the record duration read from the retrieved SLOG_intvlrec_r structure that describes the current record, while T_{start} is, of course, the statistics start time that has been estimated during the statistics initialization section. In this way, the start time as well as the end time of the currently retrieved record with respect to the start time of the statistics operation are known.

(4) The values of the parameters $B_s^{(i)}$, $B_s^{(r)}$, $B_e^{(i)}$ and $B_e^{(r)}$ are estimated as

$$B_s^{(i)} = \left\langle \frac{T'_{start}}{W} \right\rangle \tag{5}$$

$$B_s^{(r)} = T'_{start} \ \% \ W \tag{6}$$

$$B_e^{(i)} = \left\langle \frac{T'_{end}}{W} \right\rangle \tag{7}$$

$$B_e^{(r)} = T'_{end} \ \% \ W \tag{8}$$

where W is the bin width in time and $\langle X \rangle$ is the integer part of X, and more specifically, the floor part of the argument; regarding the symbol '%' it denotes the modulo operator. It is clear that the parameters $B_s^{(i)}$ and $B_e^{(i)}$ identify the time bins that contain the start time and the end time of the current record respectively, with respect to the start time of the statistics operation. On the other hand, the parameters $B_s^{(r)}$ and $B_e^{(r)}$ allows—if they are not equal to zero—the calculation of the percentage of the overlapping between the record duration and the $[B_s^{(i)}]$th and $[B_e^{(i)}]$th time bins, respectively.

In most cases, the record duration is usually much longer than the bin width in time W, and therefore, the $[B_s^{(i)}]$th and $[B_e^{(i)}]$th time bins, are separated by a large number of intermediate bins. In this case, the time bins that belong to the interval $(B_s^{(i)}, B_e^{(i)})$ are associated with a value of 1—since their overlapping with the record duration is 100%—while, the $[B_s^{(i)}]$th and $[B_e^{(i)}]$th bins are associated with the values

$$A = 1 - \frac{B_s^{(r)}}{W} \tag{9}$$

and

$$B = \frac{B_e^{(r)}}{W} \tag{10}$$

respectively. These values belong to the interval (0,1) and lead to an overlapping percentage A * 100% and B * 100% for the $[B_s^{(i)}]$th and $[B_e^{(i)}]$th bin, respectively. A typical example of this algorithm that calculates the statistics of each different record type is shown in Fig. 11.

From the above description it is obvious that after the calculation of the values of the 'bins' array, the record duration can be estimated by the equation

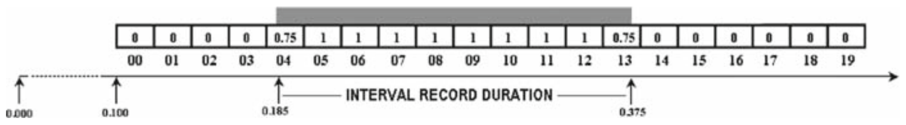


Fig. 11 The distribution of the record duration to the time bins

```
Preview Statistics:
Collect At :
Starttime = 0.02000000000000000,
Endtime =   1.60000000000000009

Preview Statistics Set 0:
rectype = 0,  intvltype = 100 label = label1
1.00,       0.01,       0.00,       0.00,       0.00,
0.00,       0.00,       0.00,       0.00,       0.00,
0.00,       0.00,       0.00,       0.00,       0.00,
0.00,       0.00,       0.00,       0.01,       1.00,

Preview Statistics Set 1:
rectype = 0,  intvltype = 101 label = label2
0.00,       0.00,       0.00,       0.00,       0.00,
0.00,       0.00,       0.00,       0.00,       0.00,
0.00,       0.00,       0.34,       0.67,       0.00,
0.00,       0.00,       0.00,       0.00,       0.00,
```

Fig. 12 The contents of a typical SLOG1 statistics section—for only two different record types—as they are printed by the slog_print utility

$$D_{rec} = W \sum_{i=0}^{N-1} Bins[i] \tag{11}$$

Even though in this equation all the time bins are involved, the record duration is going to be estimated by using only the values of the overlapping bins, since the other bins have a value of zero. This equation is valid for the simple case where the current interval record is restricted to a single time frame. On the other hand, if the record is extended to more than one frames, the duration of the record inside a frame can be estimated by using in the above equation only the time bins associated with the frame under consideration.

A typical example of the statistics of a SLOG1 file as they are printed by the slog_print MPICH utility are shown in Fig. 12.

5.7 The SLOG1 File Creation Algorithm

The SLOG1 file creation algorithm is defined as a two-phase process that creates the file in two passes. In the first pass the records are processed forward in time and they are stored in the four sectors of the file with respect to the values of the 'start time' and the ‘end time’ fields. In the second pass, the records are processed backwards in time and a set of pseudo-records are stored in the file to help the display of the frame contents to the user screen. This two-phase process is applied when the records are stored in ascending order with respect to the ‘end time’ field, while, if their ordering is ascending with respect to the ‘start time’ field, the file is created in only one phase. The description of this two-phase algorithm is presented below.

The 1st Pass of the Algorithm In the first pass of the algorithm, the four sectors of the file—described by the symbols CUR, INC, PAS, and OUT for the current, the incoming, the passing through, and the outgoing sectors, respectively—are allocated and set to their null values. An additional temporary (TMP) sector is allocated and set to the null value, too. In the next step, the file records are inserted one after the other to the INC sector. When this sector is full of data and is not capable of keeping more records, the following steps are applied:

(1) The end time of the current frame is set to the value of the end time of the last inserted record. Regarding the start time of the current frame, it is set to the end time of the previous time frame, or to the start time of the its first record, if this is the first frame of the file.
(2) The records of the TMP sector of the current frame—that come from the PAS and the OUT sectors of the previous frame—are delivered to the PAS and OUT sectors of the current frame according to their end time value. More specifically, if the end time of the current record is less or equal to the end time of the time frame, the record will be moved to the INC sector, otherwise, the record will be moved to the PAS sector. This step is performed for each time frame except the first one, for which the TMP sector is an empty sector.
(3) The records of the CUR sector of the current frame are delivered to the INC, PAS, and OUT sectors according to the relationship between the start and the end time of that records and the corresponding fields of the current frame. There are two different cases of interest: (a) the end time of the current record is less or equal to the end time of the current frame. In this case the record is moved to the INC sector if its start time is less or equal to the start time of the current frame, or to the CUR sector, otherwise. (b) the end time of the current record is greater than the end time of the current frame. In this case the record is moved to the PAS sector if the start time of the record is less than the start time of the current frame or to the OUT sector otherwise.
(4) Finally, in the last step, the records of the PAS and OUT sectors are moved to the TMP sector of the next time frame—to be processed in the same way—the current contents of the SLOG1 file are stored to hard disk, and the INC, PAS, CUR, and TMP sectors are freed, since they are not needed any more.

The 2nd Pass of the Algorithm In the second pass of the algorithm, the file records are retrieved backwards from the end to the beginning of the file and they are copied to the nodes of the four sectors defined above—these sectors initially are empty. If the current frame is the first time frame of the file, the smallest start time value of TMP, OUT and INC sectors is identified and the start time of the frame is set to that value.

In the next step, the start times of the TMP records are retrieved and according to their values, these records are moved to the OUT or the PAS sector. In a more detailed description, these records are moved to the OUT sector (if they are not already there) if their start time is greater than the start time of the current frame—or they are removed from the TMP sector, otherwise. In the same way, if the start time of a TMP record is less than the start time of the current frame, the record is moved to the PAS sector—if it is not already there—otherwise, it is removed from the TMP sector. This step is performed for each time frame except the first one, for which the TMP sector is an empty sector.

Finally, in the last step, the records of the PAS and the INC sectors are moved to the TMP sector of the next time frame to be processed in the same way, while the INC, PAS, and CUR sectors are freed, since they are not needed any more.

The basic characteristic of the second pass of the algorithm is that, if there are records whose start time or the end time belong to the time duration of another frame, they are copied to the current frame in order to be drawn quickly to the user screen without needing to find them. These records are called pseudo-records and they are very common to the SLOG1 files. Another important aspect that has to be mentioned is that the previously described algorithm can be applied only if there is an overlapping (partial or total) between the time duration of the current record and the time duration of the associated time frame. Therefore the algorithm can not be applied if these time intervals are disjoint to each other. This assumption has been set to handle the basic limitation of the above algorithm, according to which, a record can be moved between the sectors of the same time frame but not between different frames.

6 The SLOG2 File Format

The SLOG2 file format allows the so-called drawable-based logging. In this logging type, the system does not keep the values of the start and the end time only, but a whole record, that allows the complete description of the drawable objects. The main advantage of the SLOG2 file format is the creation of the file in only one stage, saving thus processing time and computational resources. The concept of the pseudo-record is not used at all and a more efficient data organization algorithm is used that puts the various drawing objects to the nodes of a binary tree according to the bounding box they belong. Another very important aspect of the SLOG2 file format is the re-normalization mechanism that allows the management of the file contents in a more sophisticated way. For these reasons, the SLOG2 files are considered more robust, reliable and efficient, compared with the SLOG1 files.

The SLOG2 log files are created by an intermediate system located between the parallel application created the data and the program that displays them in the computer screen. This intermediate system is independent of these two applications. A very important difference between the SLOG1 and the SLOG2 file formats is that the display of the data in the screen for the SLOG2 files is performed only by the display program and not by that program in cooperation with the parallel application as happens in the SLOG1 file case. Another difference between these two approaches, is that the user is not required any more to identify the time interval whose data he wants to plot; he simply clicks with the mouse pointer to the point of interest on the computer screen and the calculation of that interval is done automatically by the application and without the user intervention. To achieve a good performance, the amount of data read to refresh the screen by plotting the log data associated with the calculated time interval is not arbitrary but it has a maximum value n_{max} that depends on the file size N_F and the time duration of the logging operation, T.

The SLOG2 file format is currently supported by the logging module of the MPE library [2] (the other two modules of this library is the tracing module allowing the tracing of the source code and the animation module allowing the plotting of processes output in graphics mode), the RLOG internal MPICH2 [5] logging format, and the UTE (Unified Tracing Environment) [10] of the IBM. Regarding the application that handles this file type, this is the Jumpshot-4 Java application that runs to the X Windows graphical environment as well as to Microsoft Windows.

6.1 The SLOG2 System Architecture

The fundamental unit of the SLOG2 system is the SLOG2 file creation mechanism (known as the SLOG program) that interacts with the display program providing to it a complete SLOG2 file that can be plotted immediately or a SLOG2 annotation file that cannot be used alone but in conjunction with another file containing the data to be plotted. This data file can be one of the following file types: (a) a drawable object file that contains records describing drawable objects, (b) an event file that contains time instances and durations of the application events and (c) an application file that has been created by a third party application and may contain drawable objects as well as event records. The SLOG2 annotation file can be used with any type of those files; however, only one of those files can be used. The structure of the SLOG2 system architecture is shown in Fig. 13.

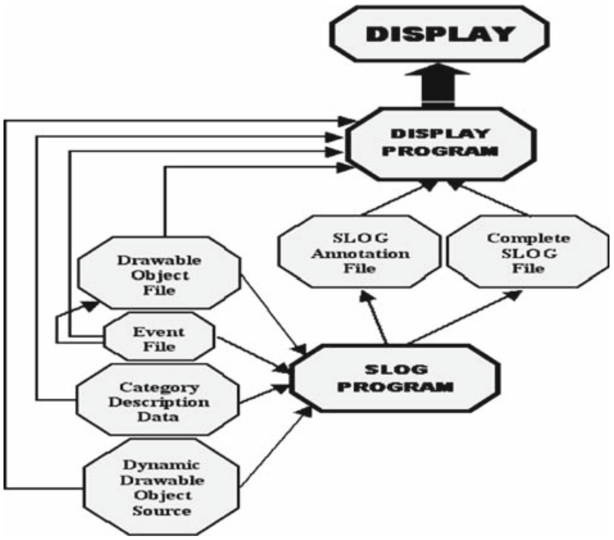


Fig. 13 The structure of the SLOG2 system architecture

The display of the SLOG2 file contents to the user screen is carried out by the display program; this program must provide at least the following functionality: (a) a set of time lines to allow the plotting of data in a Gantt diagram fashion (b) a legend to describe the various drawable objects used by the program (c) a set of popup windows to display the parameter values of the drawable objects (d) the drawable objects themselves with the definition of their property values. These properties are the legend text, the coordinates (x,y) in the computer screen, the object category, and the start and the end time of its appearance. From all these properties the most important one is the object category, since it determines the shape of the drawable object used in any case. For example, a point to point operation is plotted as an arrow, while a collective operation is shown by drawing a polygon.

The SLOG2 file format defines two types of drawable objects, the primitive objects that are considered as the building blocks for more complex structures, and the composite objects that are constructed by putting together a set of primitive objects. There are three types of primitive objects, namely, the arrows, the states, and the events. The arrows and the states are identified by two points with coordinates

$$(x, y) = (timeStamp, timeLine) \tag{12}$$

while the events are described by only one such point.

The last important aspect associated with the SLOG2 file format is the way of interpretation of the role of the vertical axis of the time space diagram: in contrast with the horizontal axis that always represents the time duration, the vertical axis can be interpreted in many different ways. In the simplest case, this axis represents the rank of the processes of the parallel application, while, in other more complicated cases the y coordinate can be used to describe a triple in the form (node id, cpu id, thread id). These three parameters have to be combined to a simple positive 32-bit integer, a procedure that can be done by using the coordinate mapping mechanism provided by the SLOG2 standard for this reason. There are three different such maps: (a) no coordinate map: in this case no mapping is carried out and the y coordinate represents the process rank (b) bit range map: in this case the y coordinate is derived by the concatenation of small binary fields with the total bit length to be no greater that 32 bits (c) general map: in this case the y coordinate is a pointer of an array that can be used to identify the quadruple (process id, thread id, cpu id, node id).

6.2 The SLOG2 Annotation File

The basic idea behind the SLOG2 file format is the concept of the bounding rectangle and the drawing of only those records that belong to this region. Each one of these records is represented by a small box and it is completely identified by the time line of the associated process and the time interval $[t_s, t_e]$ defined by its start time and end time, t_s and t_e, respectively. In the following description these records are assumed to be uniformly distributed to the time interval T defined by the duration of the SLOG2 file records, and stored in ascending order with respect to the value of their end time, t_e.

The SLOG2 annotation file is characterized by the existence of a binary tree defined in the following way: the time duration T of the whole file is divided continuously by a factor of 2, and the resulting time intervals are associated with the nodes of the binary tree, in such a way, that the duration of the intervals of the nodes of some level—it is obvious that the nodes of the same level are associated with time intervals with the same duration—to be twice as big the duration of the intervals of the nodes associated with the lower level and the half of the duration of the intervals associated with the nodes of the higher level. In other words, the root node is associated with the time interval $[0, T]$, the two nodes of the first level corresponding to the intervals $[0, T/2]$ and $[T/2, T]$ each one of them has a duration of $T/2$, the four nodes of the second level are associated with the time intervals $[0, T/4]$, $[T/4, T/2]$, $[T/2, 3T/4]$ and $[3T/4, T]$ each one of them has a duration of $T/4$, end so on.

Even though such a binary tree could have an infinite number of levels, this does not happen, since, if a record duration is less or equal to the value of a specific time interval ΔT_{min} it is supposed to belong to the one of the leafs of the tree. It can easily be proven that this time interval is given by the equation

$$\Delta T_{min} = \frac{T}{2} 2^{-\frac{ln(N_F/n_{max})}{ln(2)}} \tag{13}$$

where N_F is the size of the SLOG2 file and n_{max} is the amount of data that have to be read from the file in order the user screen to be refreshed. The division by 2 is due to the fact that the preview program reads two leafs in one step, since each record time interval with a duration equal to ΔT_{min} intersects at most the time intervals associated with two nodes. Regarding the records of the log file, they are placed to the node of the binary tree associated with the smallest time interval to which they belong.

The binary tree approach described above makes the search of the records to be displayed in the computer screen a binary search. Actually, for each record defined by the interval $[t_1, t_2]$, the tree nodes corresponding to the intervals $[t_1, (t_1 + t_2)/2]$ and $[(t_1 + t_2)/2, t_2]$ can be easily identified. Furthermore, if the user wants to display a record containing a specific time instance, the search procedure will be restricted only to those nodes whose time intervals contain that instance. In case of Fig. 14, the time instance t belongs to the time interval associated with the node 09 as well as to the time intervals of the nodes 00, 01, and 04. Therefore the identification of this record can be very easily performed by restricting it to the path formed by the nodes 00, 01, 04, and 09.

It is important to say at this point that the binary tree structure described above characterizes not only the structure of the SLOG annotation files but also the structure of the complete SLOG2 files, too. The main differences between these two file types is that in the complete SLOG2 files the binary tree contains all the necessary information for the display of their contents in the computer screen, while, the annotation file tree contains all the file records except those associated with the tree leaf nodes. Furthermore, the tree nodes contain the positions of the tree leafs with respect to their positions in another file as well as the method used for their access.

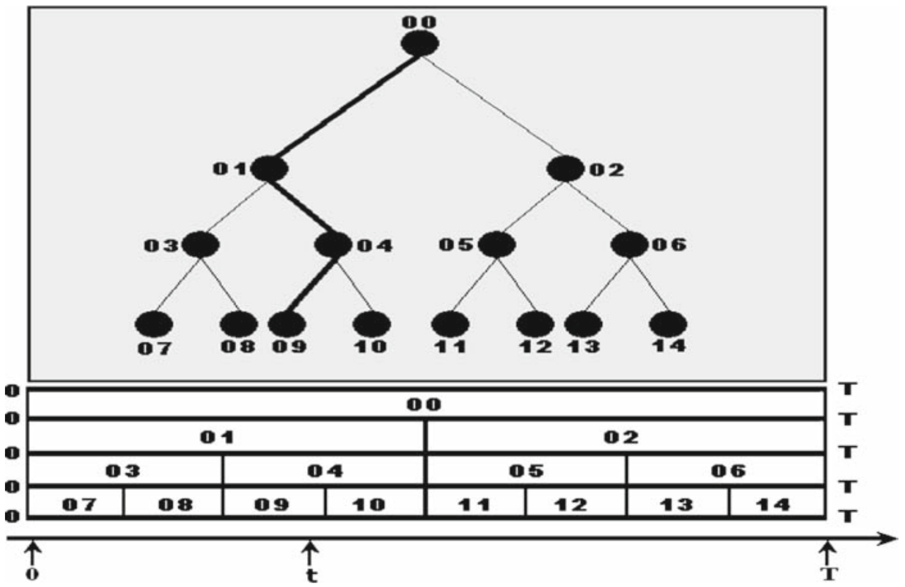


Fig. 14 The search of a time instance to the binary tree of the SLOG file

6.3 The SLOG2 File Creation Algorithm

The SLOG2 file creation algorithm is based on the assumption that the file records are stored in ascending order with respect to the value of their end time, t_e, and that the time duration of the most of them is less than or equal to ΔT_{min}. This fact imposes the placement of those records to the leaf nodes of the binary tree. In the following description it is assumed that the binary tree has $L + 1$ levels with the root of the tree to be associated with the level 0 and the leaf nodes to corresponding to level L. For each level $0 \leq \ell \leq L$ a list R_ℓ that contains the file records associated with the level ℓ is defined, and the corresponding time interval $\Delta T_\ell = 2^{-l}T$—where T is the time duration of the SLOG file—is estimated. For each node of the binary tree its data offset inside the file is stored to a directory with 2^{L-1} entries. Regarding the access of the tree nodes it is performed in a pre-order fashion—namely from the root to the leafs—to speed up the search operations.

Since the great majority of the file records has a duration less than or equal to ΔT_{min}, it is clear that the record list R_L is much longer that the lists of the other levels. For this reason, the elements of this list are stored directly to the hard disk, in contrast with the elements of the other lists that are kept in the main memory of the system. Regarding the creation of the SLOG file, it can be performed by applying the following algorithm:

– For each level $\ell (0 \leq \ell \leq L)$ the record list R_ℓ is initialized to the empty list and the time interval for this list is set to the value $[0, 2^{-l}T]$, where T is the total duration of the log file data.
– The trace file is opened.

- While the end of the file has not been reached, the following actions are performed:

  - The next record r is retrieved (with any additional information if available)
  - The levels of the binary tree are traversed from the leafs to the root and for each level the following algorithm is applied:
    - ∗ If the end time t_e of the record r is greater than the end time of the time interval ΔT_l, the record does not belong to that interval. In such a case, the R_ℓ record list is saved, and the position of the list inside the file is stored to a directory named D. In the last step, the R_ℓ list is initialized to the null list and the ΔT_ℓ is increased by the value $2^{-\ell}T$—the new value of that interval is the time duration associated with the higher binary tree level.
    - ∗ If the end time t_e of the record r is less than the end time of the time interval ΔT_ℓ, the record belongs to that interval. In this case, the record r is inserted to the record list, R_ℓ and the loop of the algorithm is terminated.
- The binary tree is traversed from the leafs to the root and for each level, ℓ, the record list, R_ℓ, is stored to the file. Furthermore, the position of the beginning of the list inside the file is stored to the D directory.
- The values of the statistic and summary data—if available—are stored to the log file.
- The contents of the directory D are stored to the log file.
- The offset of the summary data from the beginning of the file is stored to the log file.
- The number of entries of the D directory—this value is equal to 2^{L-1} is stored to the log file.

After the creation of the SLOG2 file, the contents of the time interval that the user wants to display in the computer screen, are retrieved as follows:

- The program moves to the end of the file and reads the number L of the levels of the binary tree.
- The program moves 2^{L-1} records backwards to read the record list.
- The program reads only those records whose time intervals are overlapping with the time duration specified by the user.

6.4 The SLOG2 File Format

Even though the SLOG2 files are more efficient and reliable than the SLOG1 files described above, they are under development yet, and their specifications may change in the future. For this reason, the SLOG2 format description does not include the presentation of the fundamental data structures—as in the case of the SLOG1 files—but only the structure of the data organization. Regarding the SLOG2 API, it is composed of four different groups of functions that perform the following fundamental tasks: (a) the data retrieval from a large variety of log file types such as the UTE, the ALOG, and the CLOG types (b) the retrieval of the SLOG2 file data, and (c) the creation of the SLOG2 file according to the algorithm described above. The structure of the SLOG2 file type is shown in Fig. 15 and it is composed by the following items:

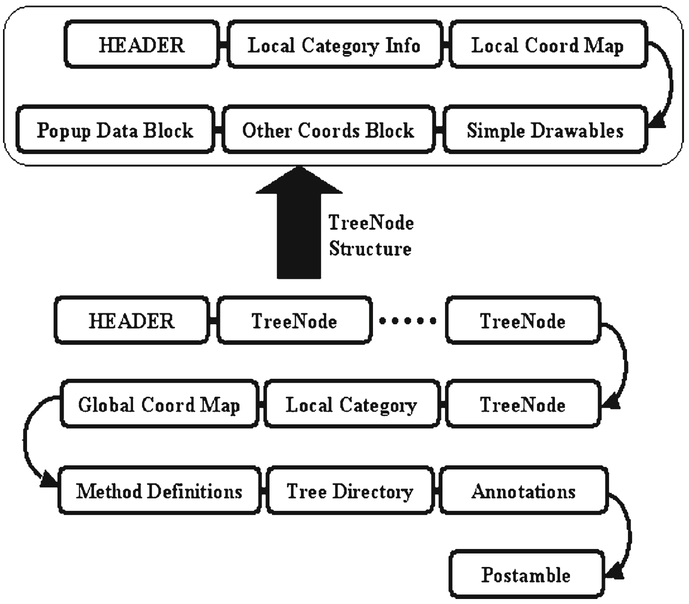
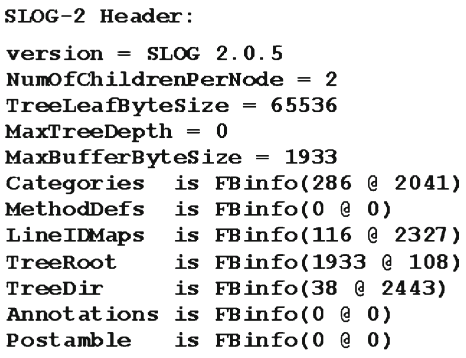


Fig. 15 The structure of the SLOG2 log file type

- Header: the header section of the SLOG2 file keeps the file version and a lot of different parameters associated with the file structure.
- Treenode: this structure represents the binary tree nodes in which the file records are stored.
- Global Category Info: this structure contains information for record categories that are stored to more than one tree nodes.
- Global Coord Map: this structure contains information about the mapping of the record parameters to the *y* coordinate value of the vertical axis, if this mapping is applied to more than one nodes.
- Method Definition: this structure contains the definition of methods used to display the SLOG2 data in the computer screen. It has not been implemented to the current version of the SLOG API.
- Tree Directory: this field is an array structure whose cells contain the starting offset of each binary tree node inside the file, together with the start and the end time associated with that node.
- Annotation: this field is a set of strings related to various information concerning the SLOG2 file. It is stored to the end of the file in order to be modified without the recreation of the whole file.
- Postamble: this is a 5-integer field (4-byte each) that keeps the starting offset for each one of the SLOG2 file sections, namely the global category info, the global coord map, the method definition, and directory, and the annotation sections.

On the other hand, the Treenode structure used to represent a binary tree node is composed of the following fields:



```
SLOG-2 Header:

version = SLOG 2.0.5
NumOfChildrenPerNode = 2
TreeLeafByteSize = 65536
MaxTreeDepth = 0
MaxBufferByteSize = 1933
Categories  is FBinfo(286 @ 2041)
MethodDefs  is FBinfo(0 @ 0)
LineIDMaps  is FBinfo(116 @ 2327)
TreeRoot    is FBinfo(1933 @ 108)
TreeDir     is FBinfo(38 @ 2443)
Annotations is FBinfo(0 @ 0)
Postamble   is FBinfo(0 @ 0)
```

Fig. 16 The contents of a typical SLOG2 file header

- Header: the header section of the tree node contains information about offset values as well as the data compression method—if such a method is used.
- Simple Drawable: this structure keeps the parameters of the simple drawable objects such as their start and the end time, the value of the *y* coordinate of the vertical axis, and the category of the drawable object.
- Other Coords Block: this structure contains the necessary coordinate values required for the display of the object if it is enough complicated.
- Popup Data Block: this structure used for the management of the popup windows that display the parameter values of the displayed objects.
- Local Category Info: this structure contains information about the categories of the objects used only by the current node.
- Local Coord Map: this structure contains information about the coordinate mapping used only by the current node.

The contents of a typical SLOG2 file header as they are printed by the slog2_print MPICH utility, are shown in Fig. 16.

7 Portability, Conversion, and Overhead

The great advantage of the log file types presented in the previous sections is the fact that even though they have been developed by the LANS group implemented the MPICH platform for MPI, they can be used by almost all the available MPI implementations; the last version of the MPE2 library (mpe2-1.0.3p1 released on 23/11/2005) can be used with OpenMPI-1.0, MPICH-1.2.7, LAM—MPI-7.1.2 b24, MPICH2—1.0.3, AIX52's MPI and BGL's MPI. The platform allows also the conversion of a log file from one format to another. The conversions CLOG $\rightarrow$ ALOG and CLOG $\rightarrow$ SLOG are performed by the utilities clog2alog and clog2slog called from the command line, while the conversions CLOG $\rightarrow$ SLOG2, CLOG2 $\rightarrow$ SLOG2 (CLOG2 is a newer CLOG format), RLOG $\rightarrow$ SLOG2 and UTE $\rightarrow$ SLOG2 are performed by the Jumpshot-4 application. These files are also supported by the IBM PE (Parallel Environment) library for AIX 5L and Linux operating systems, that provides similar capabilities such as the ability to convert TRACE UTE files to SLOG2 Jumpshot-4 compatible files.

Besides the portability of the log file types, the overhead of the logging procedure is a very important task that has to be discussed. Generally speaking, the cost of keeping trace records for specific program actions, is composed by three parts: (a) the cost of testing whether an event is enabled, (b) the cost of calling the event insertion to the trace buffer routine, and (c) the cost of wrapper routine in the tracing library. In typical situations—a five-word trace record and a modern computing node—this cost is fairly small (a small fraction of one microsecond). In the general case, the reduction of the overhead is based on local operations; in the case of the CLOG file creation the log files of each process are stored to a file of the local file system and after the completion of the operation they are merged to a file that belong to the file system of the process with rank $R = 0$. If the logging operation is characterized by user markers described by user-specified strings, the tracing library assigns an identifier to each one of those strings without any cross-task communication. However, since the MPI application is composed by multiple tasks there is no guarantee that the same identifier is returned for the same marker string. For this reason the utility that converts the trace files to interval files, re-assigns a unique identifier to each user-defined marker string, ensuring thus that the same identifier is used for the same string for all subsequent performance analysis.

The last issue discussed in this section is the size of the log files the applications can handle. In the case of the old file types, the size issue is very important. For example, the Jumpshot-2 application that reads CLOG files can handle files up to 4 MB. For larger files the performance of the application starts to decrease, while, after the limit of 10MB the application is very possible to hung up (however, Jumpshot-2 is much more efficient than its predecessor Jumpshot-1). These problems are not appear to the scalable log file formats SLOG-1 and SLOG-2 where the time needed to read a data section is independent of the file size and the position of the section inside the file. This feature allows the Jumpshot-3 and Jumpshot-4 applications that handle these file types to read log files with size of several gigabytes.

8 Conclusions and Future Work

The scope of this review paper was the detailed description of the most important log file types used for the event tracing of the parallel applications. The file types studied here was the old ALOG and CLOG files—for the sake of completeness—and the new efficient file formats SLOG1 and SLOG2. For each file type its structure and its attributes are presented both with the algorithms used for their creation.

Future work in this area includes the improvement of the log file formats presented so far as well as the development of new tools allowing the efficient manipulation of their contents. These new formats are the CLOG2 and a newer SLOG2 format. Regarding the new tools, the most important of them is the slog2filter command line utility that removes undesirable categories of drawable objects as well as alters the slog2 file structure. A variation of the slog2filter utility is the slog2updater program that reads in the older SLOG-2 file format (version 2.0.5), and writes out the latest newer one (version 2.0.6).

The development process is not restricted only to the formats of the various log files; the MPE library is continually improved with new features. The latest released version is now identified by the name MPE2 and besides the three known modules mentioned in a previous section (i.e. the tracing, the animation, and the logging module) includes a forth module—the collective and data type checking library—that checks the argument consistency for MPI collective calls. The SLOG2 SDK is also enriched by new functions and an improved API is used for the implementation of the various log file manipulation functions.

References

1. Asbury, R., Wrinn, M.: MPI tuning with Intel Trace Analyzer and Intel Trace Collector. In: IEEE International Conference on Cluster Computing, Tutorial Section (2004)
2. Chan, A., Gropp, W., Lusk, E.: User's Guide for MPE—Extensions for MPI Programs. Argonne National Laboratory, Mathematics and Computer Science Division, Technical Report ANL/MCS-TM-ANL-98 (1998)
3. Chan, A., Gropp, W., Lusk, E.: Scalable Log Files for Parallel Program Trace Data (DRAFT). Argonne National Laboratory, Technical Report (2000)
4. CLOG files documentation found in source file clog_merge.c of the MPE distribution
5. Gropp, W., Lusk, E., Ashton, D., Buntinas, D., Butler, R., Chan, A., Ross, R., Thakur, R., Toonen, B.: MPICH2 Users's Guide. Argonne National Laboratory, Mathematics and Computer Science Division, Technical Report (2008)
6. Hao, M.C., Karp, A.H., Waheed, A., Jazayeri, M.: VIZIR: an integrated environment for distributed program visualization. In: Proceedings of the 3rd International Workshop on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS '95, pp. 288–292, Durham N. Carolina (1995)
7. Heath, M.T., Finger J.E.: ParaGraph—A Performance Visualization Tool for MPI. On line document found in URL http://www.csar.uiuc.edu/software/paragraph/. Accessed 31 Aug 2003 (2003)
8. Herrarte, V., Lusk, E.: Studying Parallel Program Behavior with Upshot. Argonne National Laboratory, Technical Report ANL 91/15 (1991)
9. Hondroudakis, A., Shanmugam, K., Procter, R.: Performance evaluation and visualization with VISPAT. In: Proceedings of Parallel Computing Technologies (PaCT 1995), pp. 180–185 (1995)
10. IBM Corporation: IBM Parallel Environment for AIX 5L, Operation and Use, Vol. 2, Version 4, Release 2 (2005)
11. Karrels, E., Lusk, E.: Performance analysis of MPI programs. In: Dongarra, J., Tourancheau, B. (eds.) Proceedings of the Workshop on Environments and Tools For Parallel Scientific Computing, pp. 195–200 (1994)
12. Malony, A.D., Hammerslag, D.H., Jablonowski, D.J.: Traceview—a trace visualization tool. IEEE Software **8**(5), 19–28 (1991)
13. Pacheco, P.: Parallel Programming with MPI. Morgan Kaufmann Publishers Inc. (1997)
14. Wu, C.E., Bolmarcich, A., Snir, M., Wootton, D., Parpia, F., Chan, A., Lusk, E., Gropp, W.: From trace generation to visualization—a performance framework for distributed parallel systems. In: Proceedings of SC2000: High Performance Networking and Computing (2000)
15. Zaki, O., Lusk, E., Gropp, W., Swider, D.: Toward scalable performance visualization with jumpshot. High Perform. Comput. Appl. **13**(2), 277–288 (1999)