FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Technical Report

EPILOG Binary Trace-Data Format (Version 1.1)

Felix Wolf, Bernd Mohr*

FZJ-ZAM-IB-2004-06

May 2004

(last change: 03.05.2004)

(*) University of Tennessee

EPILOG

Binary Trace-Data Format

Version 1.1 / May 3, 2004

Felix Wolf, Bernd Mohr

Contents

1 Introduction

The development of parallel applications is a very complex and expensive process. This is essentially a result of the close relationship among the algorithm to be implemented on the one hand and the properties of the target platform in conjunction with the employed programming model on the other hand. Only when all three components of a parallel solution fit together in the right way, the application is able to achieve the desired performance.

A distinctive characteristic of modern SMP-cluster architectures is the complex hierarchical structure of their hard- and software components, which enables the use of different programming models such as MPI [5], OpenMP [6], or even the combination of the two in the same application. Naturally, the performance optimization of such applications is very complicated and involves incremental performance tuning through successive observations and code refinements. A critical step in this

procedure is to locate and explain inefficient performance behavior based on the performance data that have been collected. The quality of the resulting explanations depends to a large extent on the informative value of these data.

Event tracing provides a very fine grained view of the performance behavior of parallel applications. In contrast to pure execution-time profiling, event tracing preserves the temporal and spatial order of individual events, which may indicate the presence of certain performance properties in an application. The success of many powerful performance tools such as VAMPIR [1] or Nupshot [4] proved the usefulness of event tracing for message-passing applications.

The EPILOG (Event Processing, Investigating, and Logging) binary trace data format has been designed to extend the scope of event tracing to SMP-cluster architectures by providing a uniform data representation suitable for MPI, OpenMP, and hybrid applications. EPILOG maps events onto their location within the hierarchical hardware as well as to their process and thread of execution. It supports storage of all necessary source-code and call-site information, recording of performance metrics, such as hardware counters, and marking of collectively executed operations for both MPI and OpenMP. In addition to clusters of SMP nodes, target systems also can be meta-computing environments as well as more traditional non-cluster or non-SMP systems.

This document contains a complete specification of the EPILOG data format. It is intended for both the design of instrumentation systems that create event traces as output as well as the design of performance tools that use event traces as input. The remainder of the document is organized as follows. Section 2 deals with the layout of basic data types. Section 3 explains the overall structure of an EPILOG file, whereas Sections 4 and 5 present the individual record types in detail. Finally, Section 6 defines the symbolic constants used in the preceding sections.

2 Data Types

EPILOG defines three unsigned integer types, one floating point type and strings:

- `elg_ui1`: unsigned integer of 1 byte size
- `elg_ui4`: unsigned integer of 4 bytes size
- `elg_ui8`: unsigned integer of 8 bytes size
- `elg_d8`: IEEE 754 double precision floating point of 8 bytes size
- `elg_str`: zero terminated sequence of 1 byte ASCII characters

Additionally, EPILOG allows the creation of fixed sized vectors of arithmetic types. E.g. '`elg_ui4 vec[3]`' specifies a sequence of three `elg_ui4` values.

Finally, EPILOG defines two symbolic constants indicating the byte order of arithmetic types with the usual semantics:

- ELG_LITTLE_ENDIAN
- ELG_BIG_ENDIAN

3 File Structure

An EPILOG trace-data file consists of a header followed by a sequence of records. The header (Fig. 1) consists of the zero terminated string "EPILOG" followed by two bytes containing the major and minor EPILOG version number and another byte indicating the byte order used for arithmetic types, that is, its value is equal to one of the symbolic constants defined in the previous section.

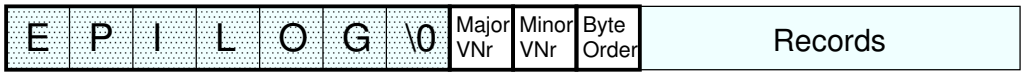

Figure 1: The EPILOG file structure.

Each record (Fig. 2) consists of the record header followed by the record body. The header contains two bytes. The first byte contains the length of the record body in bytes without these two leading bytes. The second byte contains the record type.

Figure 2: The EPILOG record structure.

EPILOG distinguishes between definition records and event records. Definition records define identifiers for objects to be referenced by other records. In particular, event records can be kept small by referencing certain objects instead of specifying these objects as part of the event record. Such objects may be source-code regions or file names. Event records represent run-time events and always contain a location identifier as well as a time stamp. A record is of a certain record type if the value of its type byte is equal to a symbolic constant with the same name as the record type.

In order to ease the processing of EPILOG trace-data files, it is recommended to place all definition records directly after the trace-file header; that is, the first event record should follow the last definition record (Fig. 3). In any case, the end of definition records is marked by a special termination record.

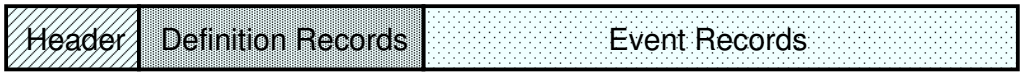

Figure 3: The EPILOG record order.

4 Definition Records

Definition records deal with the following entities of a parallel application:

- Strings
- Locations
- Source-code entities such as regions and files
- Performance metrics
- MPI communicators

EPILOG uses *identifiers* to reference objects such as source-code regions from other records. Identifiers are integer numbers of type `elg_ui4`. If an identifier has `ELG_NO_ID` ($= 2^{32} - 1$) as its value, it does not refer to an object; that is, the expected information is not present. The objects referenced by an identifier are specified in a definition record. The numeric value of an identifier should not have a specific meaning so that consistently changing identifier values should not modify the semantics of an EPILOG file. The only exceptions are process and thread identifiers which refer to particular objects of the programming models.

Sometimes, identifiers are used without defining them in a definition record (e.g., a process identifier). In this case, the identifier is only used to distinguish an object from others of the same type.

To ease the processing of EPILOG files, the identifiers used for a certain type of objects must range from 0 to $n - 1$, where n is the total number of different objects having that type.

4.1 Strings

Strings are necessary to assign a name or a description to an object. All record types that involve names or other descriptions do not contain the corresponding strings directly. Instead, they contain an identifier pointing to a string defined in a separate string-definition record.

Since the length of strings may exceed the maximum record body length, it is possible to use optional extension records, which must immediately follow the string definition record to which they refer.

ELG_STRING

This record defines an identifier `strid` for a zero terminated `string`. If `string` does not fit into one record; that is, if its length including the terminating zero exceeds 250 bytes, additional `ELG_STRING_CNT` records can be used. In this case, `cntc` contains the number of continuation records, otherwise `cntc` is zero. The

terminating zero of strings that are defined using more than one record is located in the last continuation record. The identifier can be used by subsequent records to reference the defined string.

ELG_STRING		
elg_ui4	strid	string identifier
elg_ui1	cntc	number of continuation records
elg_str	str	string

ELG_STRING_CNT

According to the cntc field of the preceding ELG_STRING record this record contains either the zero terminated rest of the string or a substring, which is continued in the immediately following ELG_STRING_CNT record.

ELG_STRING_CNT		
elg_str	str	continued string

4.2 Locations

A *location* identifies a control flow carrying an activity of the executing application. It is described by a tuple $(machine, node, process, thread)$. The different location coordinates are given by identifiers ranging from 0 to $n_i - 1$, where n_i is the maximum number of different values for coordinate i.

Machine and node characterize the physical part of a location. A machine is described by a number of nodes. A node is described by a number of CPUs. A machine identifier is globally unique, whereas a node identifier is only unique with respect to its machine.

Process and thread characterize the programming-model–related part of a location. A process identifier is globally unique, whereas a thread identifier is only unique with respect to its process. For MPI/OpenMP applications, the process identifier is equal to the MPI rank in MPI_COMM_WORLD and the thread identifier is equal to the OpenMP thread number. OpenMP nested parallelism is currently not supported.

EPILOG requires each machine to have at least one node and each process to have at least one thread. So machines that can not be divided into nodes are considered to have one node making up the entire machine. And also processes without threads are considered to have one thread making up the whole process. The identifier of these pseudo nodes and threads should be zero.

It is possible to assign a name to parts of a location, e.g. to a machine or a process. The names neither have to be unique nor they are required. For example, they can be used to assign roles to processes or to link a machine to a physical host.

ELG_MACHINE

This record assigns a number of nodes and an optional name to a machine identifier. The number of nodes is the maximum number of nodes available on this machine to an arbitrary application. If this number cannot be determined, a number greater than or equal to the number of nodes actually used can be chosen. If the machine has no name, the identifier `mnid` should be set to `ELG_NO_ID`.

ELG_MACHINE		
elg_ui4	mid	machine identifier
elg_ui4	nodec	number of nodes
elg_ui4	mnid	machine-name identifier or ELG_NO_ID

ELG_NODE

This record assigns a number of CPUs and an optional name to a node identifier. The number of CPUs is the maximum number of CPUs available on this node to an arbitrary application. If this number cannot be determined, a number greater than or equal to the number of CPUs actually used can be chosen. The machine to which the node belongs is specified by the machine identifier `mid`. If the node has no name, the identifier `nnid` should be set to `ELG_NO_ID`. The field `cr` specifies a node's clock rate. This information might be useful if the event trace contains any counted cycles as part of the event records. If the clock rate is not needed, this field can be set to zero.

ELG_NODE		
elg_ui4	nid	node identifier
elg_ui4	mid	machine identifier
elg_ui4	cpuc	number of CPUs
elg_ui4	nnid	node-name identifier or ELG_NO_ID
elg_d8	cr	number of clock cycles per second

ELG_PROCESS

This record assigns a name to a process. For example, one process can be named *server*, whereas another processes can be named *client*. This record is optional; that is, a process identifier can be used without defining a name.

ELG_PROCESS		
elg_ui4	pid	process identifier
elg_ui4	pnid	process-name identifier or ELG_NO_ID

ELG_THREAD

This record assigns a name to a thread. For example, one thread of a process can be named *master*, whereas another thread of that process can be named *worker*. This

record is optional; that is, a thread identifier can be used without defining a name. The process to which the thread belongs is specified by the process identifier `pid`.

ELG_THREAD		
elg_ui4	tid	thread identifier
elg_ui4	pid	process identifier
elg_ui4	tnid	thread-name identifier or ELG_NO_ID

ELG_LOCATION

This record assigns a 4-tuple of coordinates to a location identifier `lid`. The meaning of these coordinates is as described above. The location identifier can be used by subsequent records to reference the defined location.

ELG_LOCATION		
elg_ui4	lid	location identifier
elg_ui4	mid	machine identifier
elg_ui4	nid	node identifier
elg_ui4	pid	process identifier
elg_ui4	tid	thread identifier

4.3 Source-Code Entities

ELG_FILE

This record defines an identifier `fid` for a source file. The file name is referenced by `fnid`. The file identifier can be used by subsequent records as a reference to this file.

ELG_FILE		
elg_ui4	fid	file identifier
elg_ui4	fnid	file-name identifier

ELG_REGION

This record defines an identifier `rid` for a code region. The region name is referenced by `rnid`. The region identifier can be used by subsequent records as a reference to this region. The source-code location of the region is specified by `fid`, `begln`, and `endln`. If the source-file information is not available, `fid` is set to `ELG_NO_ID`. Missing line numbers are indicated by the value `ELG_NO_LNO`. It is possible to provide an optional region description using an additional string identifier `rdid`.

ELG_REGION		
elg_ui4	rid	region identifier
elg_ui4	rnid	region-name identifier
elg_ui4	fid	source-file identifier or ELG_NO_ID
elg_ui4	begln	begin line number
elg_ui4	endln	end line number
elg_ui4	rdid	region-description identifier or ELG_NO_ID
elg_ui1	rtype	region type

The last data field `rtype` specifies a region type. The region type is used to distinguish between different syntactical kinds of regions such as functions or OpenMP constructs. It is specified by using one of the following symbolic constants:

- ELG_FUNCTION
- ELG_LOOP
- ELG_USER_REGION
- ELG_OMP_PARALLEL
- ELG_OMP_LOOP
- ELG_OMP_SECTIONS
- ELG_OMP_SECTION
- ELG_OMP_WORKSHARE
- ELG_OMP_SINGLE
- ELG_OMP_MASTER
- ELG_OMP_CRITICAL
- ELG_OMP_ATOMIC
- ELG_OMP_BARRIER
- ELG_OMP_IBARRIER (implicit barrier)
- ELG_OMP_FLUSH
- ELG_OMP_CRITICAL_SBLOCK
- ELG_OMP_SINGLE_SBLOCK
- ELG_UNKNOWN

The last one can be used if an instrumentation system is not able to provide a region type.

ELG_CALL_SITE

This record defines an identifier `csid` for a call site. A call site is a source-code location where the control flow may move from one region to another region. It does not necessarily to be a function call site, instead, it can also be a loop entry, where the control flow may move from the enclosing region to the loop region. The call-site identifier can be used by subsequent records as a reference to this call site. The source-code location of the call site is specified by `fid` and `lno`. The region to be entered (i.e., the callee) is specified by `erid`, while the region to be left (i.e., the caller) is specified by `lrid`. The latter one is optional.

ELG_CALL_SITE		
elg_ui4	csid	call-site identifier
elg_ui4	fid	source-file identifier
elg_ui4	lno	line number
elg_ui4	erid	region identifier of the region to be entered
elg_ui4	lrid	region identifier of the region to be left or ELG_NO_ID

4.4 Performance Metrics

Frequently, it is desirable to include additional performance metrics into a trace file. For example, hardware performance counters have been proved to be a useful diagnostic tool in assessing the performance of microprocessors as well as applications running on them. Libraries, such as PAPI [3] and PCL [2], provide uniform access to these counters and simplify the design of performance tools based on them. In addition, for performance analysis it might also be useful to consider system parameters, such as the amount of memory allocated at a time.

The EPILOG trace-data format supports the recording of different performance metrics. These values may be recorded as part of the `ELG_ENTER`, `ELG_ENTER_CS`, and `ELG_EXIT` event records. EPILOG provides a definition-record type to declare a performance metric and a definition-record type to specify the order in which the corresponding values will later appear in event records.

The values of counters usually refer to a time interval during which the counting took place, whereas the values of system parameters, such as size of memory allocation, often refer to a point in time. EPILOG supports both values referring to an interval as well as values referring to a single point in time.

Frequently, application developers are also interested in occurrences-per-time ratios (e.g., instructions per second). Although EPILOG supports the storage of these rates in the same way it does support the storage of plain counters, the computation of rates at run time may introduce undesirable overhead, which can be avoided in the presence of timestamped events. Instead, rates should be always computed by converting a plain counter representation into a rate representation offline.

ELG_METRIC

This record defines an identifier `metid` for a metric, whose values may be stored in event records. It specifies the metric's name `metnid`, an optional more detailed description `metdid`, and the metric's data type `metdtype`. Whereas the name should indicate the kind of metric (e.g., floating-point instruction), the description may be used, for example, to inform about platform-specific characteristics. Each metric is stored in an 8 byte data type - either `elg_ui8` or `elg_d8`. To specify which data type is used for a particular metric, the `metdtype` field may carry one of the following two symbolic constants:

- `ELG_INTEGER`
- `ELG_FLOAT`

The flag `metmode` specifies whether the values to be measured represent a number of event occurrences, a number of event occurrences per time (i.e., a rate), or the current value of a system parameter. Whereas the first two modes refer to a time interval, the last mode refers to a point in time. The flag `metmode` may carry one of the following symbolic constants:

- `ELG_COUNTER`: number of event occurrences
- `ELG_RATE`: number of event occurrences per time
- `ELG_SAMPLE`: current value of a system parameter

In the case of the first two modes, the flag `metiv` can be used to specify the interval a value represents. Specifying the interval since start of measurement will result in monotonically increasing values.

- `ELG_START`: interval since start of measurement on a location
- `ELG_LAST`: interval since last measurement on a location
- `ELG_NEXT`: interval to next measurement on a location

Since the availability and characteristics of performance counters and system parameters may change as new microprocessor architectures emerge, EPILOG does not predefine any metrics to be recorded in an event trace.

ELG_METRIC		
elg_ui4	metid	metric identifier
elg_ui4	metnid	metric-name identifier
elg_ui4	metdid	metric-description identifier or ELG_NO_ID
elg_ui1	metdtype	metric data type
elg_ui1	metmode	metric mode
elg_ui1	metiv	time interval to which the metric refers

However, to encourage tool interoperability, EPILOG recommends to use predefined names for commonly used hardware counters. Rates based on these counters can be specified using the predefined counter name while having the flag `metmode` set to `ELG_RATE`.

Memory hierarchy

- `L<`*n*`><`*d*`>_ACCESS`: level *n* cache accesses
- `L<`*n*`><`*d*`>_READ`: level *n* cache reads
- `L<`*n*`>_WRITE` : level *n* cache writes
- `L<`*n*`><`*d*`>_HIT`: level *n* cache hits
- `L<`*n*`><`*d*`>_READ_HIT`: level *n* cache read hits
- `L<`*n*`>_WRITE_HIT`: level *n* cache write hits
- `L<`*n*`><`*d*`>_MISS`: level *n* cache misses
- `L<`*n*`><`*d*`>_READ_MISS`: level *n* cache read misses
- `L<`*n*`>_WRITE_MISS`: level *n* cache write misses
- `TLB<`*d*`>_HIT`: translation-lookaside-buffer hits
- `TLB<`*d*`>_MISS`: translation-lookaside-buffer misses

These counter types denote cache and translation-lookaside-buffer accesses. `<`*n*`>` can be replaced by the cache-level number, `<`*d*`>` either by "`_I`" for instruction, "`_D`" for data accesses, or it can be omitted in cases where the distinction is irrelevant.

Instructions

- `INSTRUCTION`: instructions completed
- `INTEGER`: integer instructions completed
- `FLOATING_POINT`: floating-point instructions completed
- `LOAD_STORE`: load or store instructions completed
- `LOAD`: load instructions completed
- `STORE`: store instructions completed
- `COND_STORE`: conditional store instructions completed
- `COND_STORE_SUCCESS`: successful conditional stores
- `COND_STORE_UNSUCCESS`: unsuccessful conditional stores
- `COND_BRANCH`: conditional branches
- `COND_BRANCH_TAKEN`: conditional branches taken
- `COND_BRANCH_NOTTAKEN`: conditional branches not taken
- `COND_BRANCH_PRED`: conditional branches correctly predicted

- COND_BRANCH_MISPRED: conditional branches mispredicted

Cycles

- CYCLES: cycles on behalf of the process/thread
- ELAPSED_CYCLES: elapsed cycles

Idle Units

- INTEGER_UNIT_IDLE: cycles integer units are idle
- FLOAT_UNIT_IDLE: cycles floating-point units are idle
- BRANCH_UNIT_IDLE: cycles branch units are idle
- LOADSTORE_UNIT_IDLE: cycles load-store units are idle

Stalls

- STALL_MEMORY_ACCESS: cycles stalled waiting for memory access
- STALL_MEMORY_READ: cycles stalled waiting for memory read
- STALL_MEMORY_WRITE: cycles stalled waiting for memory write

The metrics that have been defined in an event trace using this record will later appear in event records in the order given by the metric identifiers. That is, the first metric value in an event record corresponds to the metric whose `metid` is zero, the second metric value in an event record corresponds to the metric whose `metid` is one, etc.. Note that all the metrics defined in an event trace need to consistently appear in all ELG_ENTER, ELG_ENTER_CS, and ELG_EXIT event records in the trace.

4.5 MPI Communicators

ELG_MPI_COMM

This record defines an identifier `cid` for an MPI communicator. The communicator identifier can be used by subsequent records as a reference to this communicator. The communicator is characterized by its MPI group and an optional name. The group is given as a bit string `grpv[]` consisting of `grpc` bytes. The nth bit of the bit string being set indicates that the process n; that is, the rank n of MPI_COMM_WORLD belongs to this group. If the size of MPI_COMM_WORLD is not divisible by eight, the unused bits of the last byte are ignored.

ELG_MPI_COMM		
elg_ui4	cid	communicator identifier
elg_ui4	grpc	size of the bit string in bytes
elg_ui1	grpv[grpc]	bit string defining the group

Future versions of EPILOG may include source-code information related to MPI communicators.

4.6 Clock Synchronization

ELG_OFFSET

In system relying only on local clocks the time stamps of event records from different locations can be adjusted based on the information contained in this record. It is temporarily used in temporary trace files containing time stamps local to a single location. A trace file with global time stamps does not contain records of this type.

It is assumed, that the local clocks differ in offset and frequency, and that the offset between two clocks can be described as a linear function of the time. Without loss of generality, one local clock is assumed to provide the global time. Knowing the offset between an arbitrary local clock and the global clock for two different points in time allows the definition of the global time as a function of the local time. Of course, the accuracy of this method can be increased by choosing the highest possible temporal distance between these points in time, so offsets should be measured once at program initialization and once at program termination.

A record of this type provides the `offset` between the local time and the global time at `ltime` local time. The unit of time values is always seconds.

ELG_OFFSET		
elg_d8	ltime	local time
elg_d8	offset	offset to global time = global time - local time

4.7 Number and Position of Records

ELG_LAST_DEF

This record indicates that all record following this record are event records. It has no data fields. A program collecting static information from a trace file can stop searching for definition records as soon as this record appears.

ELG_NUM_EVENTS

This record provides the number of event records contained in a trace file. It can be used to implement progress reports while processing a trace file.

ELG_NUM_EVENTS		
elg_ui4	eventc	number of event records

5 Event Records

All event records provide a location identifier and a time stamp. The time stamp is of type `elg_d8`. The unit for time stamps is always seconds. EPILOG provides records for the following kinds of events:

- Entering and leaving regions
- MPI point-to-point communication
- MPI collective communication
- OpenMP fork and join
- OpenMP parallel execution
- OpenMP lock synchronization
- Tracing events (i.e., events related to the tracing system)

When an optional data field is omitted, the following data fields are directly placed after the last used data field. So the tables only specify the relative order of the data fields and not their absolute positions. The presence of optional data fields can be derived from the total record length and - if necessary - from preceding definition records.

5.1 Programming-Model–Independent Events

The following three record types are used for entering or leaving a code region. They provide a vector `metv[]` containing the values of previously specified performance metrics. The number and order of values is specified by the number of `ELG_METRIC` records and the order of their corresponding metric identifiers. If there is no `ELG_METRIC` record, `metv[]` is omitted. But if there are any metric defined using `ELG_METRIC` records, their values have to be present in all records of these types.

ELG_ENTER

This record indicates that the program enters a code region denoted by `rid`.

ELG_ENTER		
elg_ui4 elg_d8	lid time	location identifier time stamp
elg_ui4	rid	region identifier of the region being entered
elg_ui8 | elg_d8	metv[]	metric values

ELG_ENTER_CS

This record indicates that the program enters a code region from a call site denoted by `csid`. This record should be used instead of `ELG_ENTER` if the instrumented point is the call site and not the region entry.

ELG_ENTER_CS		
elg_ui4 elg_d8	lid time	location identifier time stamp
elg_ui4	csid	call-site identifier of the call site being executed
elg_ui8 | elg_d8	metv[]	metric values

ELG_EXIT

This record indicates that the program leaves the last code region being entered.

ELG_EXIT		
elg_ui4 elg_d8	lid time	location identifier time stamp
elg_ui8 | elg_d8	metv[]	metric values

5.2 MPI-Related Events

The MPI-related events supported by EPILOG include sending a message, receiving a message, and leaving a collective operation.

ELG_MPI_SEND

This record indicates the dispatch of a message. dlid specifies the destination location of the message. cid is the identifier of the MPI communicator used for message transfer. tag is the message tag and sent contains the number of bytes sent.

ELG_MPI_SEND		
elg_ui4 elg_d8	lid time	location identifier time stamp
elg_ui4 elg_ui4 elg_ui4 elg_ui4	dlid cid tag sent	destination-location identifier of message communicator identifier message tag message length in bytes

ELG_MPI_RECV

This record indicates the receipt of a message. slid specifies the source location of the message. cid is the identifier of the MPI communicator used for message transfer. tag is the message tag. The number of bytes received can be derived from the corresponding ELG_MPI_SEND record.

ELG_MPI_RECV		
elg_ui4	lid	location identifier
elg_d8	time	time stamp
elg_ui4	slid	source-location identifier of message
elg_ui4	cid	communicator identifier
elg_ui4	tag	message tag

ELG_MPI_COLLEXIT

This record indicates that the program leaves an MPI collective operation. It is used instead of an ELG_EXIT event, when a collective operation instance is left. Therefore, it offers all data fields a simple ELG_EXIT provides. But in contrast to pure ELG_EXIT events it has additional data fields, which deliver information on the communication done during the collective operation. rlid specifies the root location of the operation. cid is the identifier of the MPI communicator used for the collective operation. sent and recvd specify the amount of bytes sent or received by the location during this particular collective-operation instance.

ELG_MPI_COLLEXIT		
elg_ui4	lid	location identifier
elg_d8	time	time stamp
elg_ui8 | elg_d8	metv[]	metric values
elg_ui4	rlid	root-location identifier of the operation
elg_ui4	cid	communicator identifier
elg_ui4	sent	bytes sent
elg_ui4	recvd	bytes received

5.3 OpenMP-Related Events

The OpenMP-related events supported by EPILOG include fork and join operations, acquiring and releasing a lock, and leaving a region executed in parallel (i.e., an OpenMP parallel construct).

ELG_OMP_FORK

This record indicates that the master thread creates a team of threads, which execute in parallel from that time on. The location of the event is always the master thread. Event records of non-master threads are only allowed to appear between an ELG_OMP_FORK record and an ELG_OMP_JOIN record, which is described below. However, records generated by non-master threads that indicate events of the tracing system as described in Section 5.4 are allowed to appear also outside fork and join records.

ELG_OMP_FORK		
elg_ui4	lid	location identifier
elg_d8	time	time stamp

ELG_OMP_JOIN

This record indicates that the non-master threads of a team finish their execution. Only the master thread continues its execution. The location of the event is always the master thread. Event records of non-master threads are only allowed to appear between an `ELG_OMP_FORK` record, which is described above, and an `ELG_OMP_JOIN` record. However, records generated by non-master threads that indicate events of the tracing system as described in Section 5.4 are allowed to appear also outside fork and join records.

ELG_OMP_FORK		
elg_ui4	lid	location identifier
elg_d8	time	time stamp

ELG_OMP_ALOCK

This record indicates that a simple or nested OpenMP lock is acquired. `lkid` is the identifier of the lock being set.

ELG_OMP_ALOCK		
elg_ui4	lid	location identifier
elg_d8	time	time stamp
elg_ui4	lkid	identifier of the lock being acquired

ELG_OMP_RLOCK

This record indicates that a simple or nested OpenMP lock is released. `lkid` is the identifier of the lock being released.

ELG_OMP_RLOCK		
elg_ui4	lid	location identifier
elg_d8	time	time stamp
elg_ui4	lkid	identifier of the lock being released

ELG_OMP_COLLEXIT

This record indicates that the program leaves an OpenMP parallel region. OpenMP parallel regions are defined in the source code using OpenMP parallel constructs. The record is used instead of an `ELG_EXIT` record, when such a parallel region instance is left. Therefore, it offers all data fields a simple `ELG_EXIT` provides.

ELG_OMP_COLLEXIT		
elg_ui4	lid	location identifier
elg_d8	time	time stamp
elg_ui8 | elg_d8	metv[]	metric values

5.4 Tracing Events

The next two record types are used, when tracing is temporarily turned off. When it is turned on again, the application must execute on the same stack level with respect to traced regions. Also all message sent during that interval must be received during that interval and vice versa. That means turning off tracing is not allowed to introduce any inconsistencies in the event trace. The last two record types are used, when the trace buffer is dumped to a file. They are placed like normal region enters and exits and contain an array of metric values analogously to their specification.

ELG_LOG_OFF

This record indicates that tracing is temporarily turned off.

ELG_LOG_OFF		
elg_ui4 elg_d8	lid time	location identifier time stamp
elg_ui8 | elg_d8	metv[]	metric values

ELG_LOG_ON

This record indicates that tracing is turned on again. It must follow a preceding ELG_LOG_OFF at the same location.

ELG_LOG_ON		
elg_ui4 elg_d8	lid time	location identifier time stamp
elg_ui8 | elg_d8	metv[]	metric values

ELG_ENTER_DUMP

This record indicates that the tracing system starts writing out its buffer.

ELG_ENTER_DUMP		
elg_ui4 elg_d8	lid time	location identifier time stamp
elg_ui8 | elg_d8	metv[]	metric values

ELG_EXIT_DUMP

This record indicates that the tracing system finishes writing out its buffer. It must follow a preceding ELG_ENTER_DUMP at the same location.

ELG_EXIT_DUMP		
elg_ui4 elg_d8	lid time	location identifier time stamp
elg_ui8 | elg_d8	metv[]	metric values

6 Symbolic Constants

This section defines the symbolic constants used in the preceding sections. All values are given as decimal numbers.

Symbolic Constants	
Byte order	
ELG_LITTLE_ENDIAN	1
ELG_BIG_ENDIAN	2
Absent information	
ELG_NO_ID	$2^{32} - 1$
ELG_NO_LNO	$2^{32} - 1$
Definition record types	
ELG_STRING	1
ELG_STRING_CNT	2
ELG_MACHINE	3
ELG_NODE	4
ELG_PROCESS	5
ELG_THREAD	6
ELG_LOCATION	7
ELG_FILE	8
ELG_REGION	9
ELG_CALL_SITE	15
ELG_METRIC	10
ELG_MPI_COMM	11
ELG_OFFSET	12
ELG_LAST_DEF	13
ELG_NUM_RECS	14
Event record types	
ELG_ENTER	101
ELG_ENTER_CS	111
ELG_EXIT	102
ELG_MPI_SEND	103
ELG_MPI_RECV	104
ELG_MPI_COLLEXIT	105
ELG_OMP_FORK	106
ELG_OMP_JOIN	107
ELG_OMP_ALOCK	108
ELG_OMP_RLOCK	109
ELG_OMP_COLLEXIT	110
ELG_LOG_OFF	201
ELG_LOG_ON	202
ELG_ENTER_DUMP	203

Symbolic Constants (cont.)	
ELG_EXIT_DUMP	204
Region types	
ELG_UNKNOWN	0
ELG_FUNCTION	1
ELG_LOOP	2
ELG_USER_REGION	3
ELG_OMP_PARALLEL	11
ELG_OMP_LOOP	12
ELG_OMP_SECTIONS	13
ELG_OMP_SECTION	14
ELG_OMP_WORKSHARE	15
ELG_OMP_SINGLE	16
ELG_OMP_MASTER	17
ELG_OMP_CRITICAL	18
ELG_OMP_ATOMIC	19
ELG_OMP_BARRIER	20
ELG_OMP_IBARRIER	21
ELG_OMP_FLUSH	22
ELG_OMP_CRITICAL_SBLOCK	23
ELG_OMP_SINGLE_SBLOCK	24
Performance metrics	
ELG_INTEGER	0
ELG_FLOAT	1
ELG_COUNTER	0
ELG_RATE	1
ELG_SAMPLE	2
ELG_START	0
ELG_LAST	1
ELG_NEXT	2

7 Revision History

This section describes the difference between revisions of the EPILOG format.

7.1 Revision 1.1

The changes include a redesign of the hardware-counter support features, support for call-site tracing, and the addition of a new definition record telling the number

of event records in a trace.

- `ELG_PCNTS` replaced by `ELG_METRIC` record
- `metv[]` field added to the `ELG_LOG_OFF`, `ELG_LOG_ON`, `ELG_ENTER_DUMP`, and `ELG_EXIT_DUMP` records.
- `cr` field added to the `ELG_NODE` record
- `ELG_CALL_SITE` record added
- `ELG_ENTER_CS` record added
- `ELG_NUM_EVENTS` record added
- Semantics of the `nodec` field in the `ELG_MACHINE` record modified
- Semantics of the `cpuc` field in the `ELG_NODE` record modified

References

[1] A. Arnold, U. Detert, and W.E. Nagel. Performance Optimization of Parallel Programs: Tracing, Zooming, Understanding. In R. Winget and K. Winget, editors, *Proc. of Cray User Group Meeting*, pages 252–258, Denver, CO, March 1995.

[2] R. Berrendorf and H. Ziegler. PCL - The Performance Counter Library: A Common Interface to Access Hardware Performance Counters on Microprocessors. Technical Report IB-9816, Forschungszentrum Jülich, October 1998. `http://www.fz-juelich.de/zam/PCL/`.

[3] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A Portable Programming Interface for Performance Evaluation on Modern Processors. *The International Journal of High Performance Computing Applications*, 14(3):189–204, 2000. `http://icl.cs.utk.edu/papi/`.

[4] E. Karrels and E. Lusk. Performance Analysis of MPI Programs. In *Proc. of the Workshop on Environments and Tools for Parallel and Scientific Computing*, 1994.

[5] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, June 1995. `http://www.mpi-forum.org/`.

[6] OpenMP Architecture Review Board. OpenMP Fortran Application Program Interface – Version 2.0, November 2000. `http://www.openmp.org`.