ORNL/TM--12125

DE93 002355

Engineering Physics and Mathematics Division

Mathematical Sciences Section

A NEW PICL TRACE FILE FORMAT

Patrick H. Worley

Oak Ridge National Laboratory
Mathematical Sciences Section
P. O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

worley@msr.epm.ornl.gov

Date Published: October 1992

Research was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy.

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Contents

1 Introduction 1
2 New format 2
  2.1 General structure of a trace record 2
  2.2 General conventions for fields 3
    2.2.1 Field representation conventions 3
    2.2.2 Field value conventions 4
  2.3 PICL/ParaGraph-specific interpretation of field values 5
    2.3.1 Record types 5
    2.3.2 Event types 7
    2.3.3 Data fields for event entry, exit, and mark records 11
    2.3.4 Data fields for event label records 17
    2.3.5 Data fields for event data descriptor records 17
    2.3.6 Data fields for event message records 17
    2.3.7 Data fields for statistics records 17
    2.3.8 Data fields for subset-definition records 19
  2.4 Header records 19
  2.5 ParaGraph requirements 20
3 Future work 21
Acknowledgments 21
A Example trace file 21
B References 25

A NEW PICL TRACE FILE FORMAT

Patrick H. Worley

Abstract

A trace file format is described that will be used in future releases of the Portable Instrumented Communication Library (PICL) and ParaGraph. The new format provides improved support for tracing and profiling PICL communication primitives and user-defined events. The new format is also easily extended and may be useful in other instrumentation packages and performance visualization tools.

1. Introduction

The Portable Instrumented Communication Library (PICL) is a subroutine library developed at Oak Ridge National Laboratory (ORNL) that implements a generic message-passing interface on a variety of multiprocessors [4], [5], [6]. Programs written using PICL routines instead of the native commands for interprocessor communication are portable in the sense that they can be run on any machine on which the library has been implemented. PICL was designed

1) to provide portability between the target multiprocessors at low overhead,

2) to provide trace data on interprocessor communications and user-defined tasks, again at low overhead to avoid significantly perturbing the behavior being observed, and

3) to be easily maintained, easily ported to new systems, and easily extended to take advantage of (new) native commands that are important for performance.

To support these design goals, PICL is implemented as a C subroutine library, with a machine independent tracing layer and a machine-dependent layer that maps directly into the native commands (when possible). PICL was made available to external users in March of 1989. Since then, development has continued and several updates have been released, adding new communication and tracing commands and ports to new multiprocessors.

Instrumentation logic was added to PICL in order to collect data for performance tuning and debugging and to collect data for a research project on modeling the performance of parallel programs [16]. To support these efforts, a software tool called ParaGraph was developed for visualizing the behavior and performance of parallel programs on message-passing multiprocessors [7], [8]. ParaGraph uses the trace information produced by PICL as input, but that is ParaGraph's only dependence on PICL. ParaGraph was made available to external users in April of 1990, and its development has also continued.

Since their releases, both PICL and ParaGraph have become popular, with sizeable user bases and interest from vendors and other research groups in supporting the packages directly. In recognition of the needs of these external users, a conservative development policy was adopted, emphasizing backward compatibility and restricting modifications to those that directly address user needs. For example, until recently, PICL's trace file format has been adequate for our needs, and has not been significantly modified since the initial design.

The current PICL trace file format is based on a format used in Dunigan's hypercube simulator mpsim [3], on the performance data collected automatically by the operating system of the nCUBE 3200 [12], and on our expectations about what type of data we would need for our performance modeling research. Early on, we were concerned about the volume of data to be collected, and we concentrated on supporting the sampling of cumulative performance

statistics. While sampling performance statistics is an important and scalable approach to measuring performance, other approaches have proved equally important. PICL trace data is used most commonly as input to ParaGraph, which requires detailed event records. Also, our performance tuning and modeling work are increasingly dependent on tracing and profiling user-defined events. While the current format provides sufficient information for both of these activites, the interpretation of the data is often "context-sensitive", requiring that multiple trace records be examined in a specific sequence in order to extract the desired information. It is also difficult to update the current trace file format to include new information, especially events corresponding to new PICL commands, without losing backward compatibility. Finally, due to ParaGraph's popularity, other software developers have implemented PICL's trace file format in their systems, or have developed programs to translate their native formats to/from the PICL format. This can be an unnecessarily complicated task because of the peculiarities of the current PICL format. These deficiencies prompted us to redesign the PICL trace file format.

2. New format

The new PICL trace file format is not designed to be a standard. In discussions on standardizing trace file formats, the consensus seems to be that a standard metaformat to describe how to interpret a trace file would be more useful than a standard trace format, given the different requirements of hardware, operating system, and application code level instrumentation [2]. Instead, the new format is designed to address the specific needs of our modeling research and of PICL and ParaGraph users. Since these requirements are common to many other instrumentation and visualization systems, we also tried to make the new format general enough to be used in other (similar) systems. (See, for example, [2], [9], [10], [13], [14], and [15].)

To emphasize the generality of the new format, it is specified in three levels of decreasing generality. Section 2.1 describes the general structure of a trace record: the number, intended use, and order of fields in a trace record. Section 2.2 describes the default representation of each field, as used in PICL and ParaGraph, and some general conventions on the values of the fields that are motivated by PICL and ParaGraph, but are not specific to them. Section 2.3 describes the PICL/ParaGraph-specific interpretation of each value for each field. Appendix A contains a sample PICL trace file generated using the new format.

2.1. General structure of a trace record

In the new format, trace records have the fields described in Table 1. Every record will have the first six fields, but if the number of data fields is zero, then the data descriptor field is eliminated.

Field name	*Meaning*
record type	type of information in trace record
event type	type of event that information is associated with
timestamp	when information was valid
processor id	processor that information is associated with
process id	process that information is associated with
number of data fields	number of other data fields associated with record and event types
data descriptor	format of other data fields
data	other data fields

Table 1: Basic structure of a trace record

The emphasis in the new format is on recording information associated with events, both system and user-defined, with every record associated with some event type. The ordering of the fields reflects the importance of the fields to an animation system like ParaGraph: what, when, where, and associated data. If the record or event type is not recognized by the system, then it can skip to the next record. But, if the system wants to read in the data, then it has enough information to do so.

Note that nothing has been said about the format of these fields or about how to interpret the values of the fields. This is partially motivated by Reed's argument against specifying data semantics in a portable trace file format, although it does not go as far as his proposal [2]. Special (header) records can be included at the beginning of a trace file to specify how to read and interpret these fields, if a standard metaformat is ever agreed upon. (See, for example, [1].) The PICL defaults for the new format are described in subsequent sections.

2.2. General conventions for fields

2.2.1. Field representation conventions

The (default) PICL format continues to be an "external", human-readable format. All fields are in ASCII character format and are separated by white space. The record type, event type, processor id, process id, and number of data fields are all integers. The data descriptor is either an integer or a variable length character string delimited by the double-quote character. The format of the data fields is indicated by the data descriptor field, as will be described in the next section. The timestamp is a floating point number whose resolution is a function of the units (e.g., seconds) and of the clock resolution of the machine on which the trace was generated. Note that while it may be useful to specify the clock resolution in a header record, it is not necessary. If some care is taken during input, then the timestamp can be read without losing resolution.

We have found a human-readable syntax to be very useful for portability, for finding and correcting errors in trace files, and for perusing the data without the aid of a visualization

system, and it does not preclude compressing the file if it is so desired. Note that PICL uses a binary format internally to save space and to avoid the overhead of type conversion during the program execution, but this is never seen by the user.

2.2.2. Field value conventions

The interpretation of the field values in the new format are all specific to PICL, but the conventions described in Table 2 may be useful to other system designers who wish to adopt the new PICL syntax.

Field	*Conventions*
record type	if ≥ 0, then a user-defined record type if < 0, then a standard or system-defined record type
event type	if ≥ 0, then a user event or a user-defined subset of events if $= -1$, then refers to all events (global information) if < -1, then a system event or a system-defined subset of events
timestamp	time in seconds.
processor id	if ≥ 0, then the id of an individual processor if $= -1$, then refers to all processors (global information) if < -1, then refers to subsets of processors
process id	if ≥ 0, then the id of a process on the specified processor if $= -1$, then refers to all processes on the specified processor (global information) if < -1, then refers to subsets of processes on the specified processor
number of data fields	number of other data fields.
data descriptor	if double-quote delimited, then a `scanf` control string otherwise, if $= 0$, then character data (`"%c"`) $= 1$, then string data (`"%s"`) $= 2$, then integer data (`"%d"`) $= 3$, then long integer data (`"%ld"`) $= 4$, then single precision floating point data (`"%f"`) $= 5$, then double precision floating point data (`"%lf"`) ≥ 6, then some other predefined data format

Table 2: Basic conventions on field values

The processor and process id fields are assumed to define the "location" of the associated event. While the usual approach is to number the processes associated with each processor sequentially starting with 0, it may also be useful to allow the process numbering to be independent of the processor, directly supporting both processor–dependent and processor–independent approaches to visualizing the performance. (Note that neither PICL nor ParaGraph currently support multiple processes on a (serial) processor. The process id field has been introduced for compatibility with systems that do support this programming model, and to allow PICL and ParaGraph a growth path. In PICL trace files using the new format, the process id field is set

either to the actual PID, i.e. the identification number assigned to the process by the native operating system, or to the value -1.)

In general, the data descriptor field will contain a **scanf** control string [11]. This is a (variable length) character string, delimited by the double-quote character, that describes the format of each subsequent data field. The control string may describe multiple data values, for example, an integer and a floating point number. In this case, each data field is defined to contain multiple data values, representing a structure defined by the control string. For simple data fields, a control string data descriptor is relatively compact. For example, the number of fields and the data descriptor for a record with data consisting of three integers would be either 3 and **"%d"**, respectively, or 1 and **"%d%d%d"**, whichever is more natural for the given record. But a control string data descriptor also supports a great deal of flexibility, allowing data of different types to be included in the same record. For complex mixes of data types, the control string may be very long. To mitigate the effect of this length on the size of the trace file, a control string can be defined in a special record and represented by an integer value in subsequent records. See §2.3 for a description of this record. As indicated in Table 2, certain of these integer values have been predefined. For example, a data descriptor field value of 2 is equivalent to the control string **"%d"**. These predefined integer values for the data descriptor field are primarily meant to be examples, and can be redefined by using the special record type.

For most fields, the value -1 is designated to be a wildcard, and has special interpretations. A wildcard field value is used most often to denote that the associated information is "global", applying to all events, processors, or processes. Depending on the specific use, the wildcard can also be interpreted as a "do not care" or a "do not know" value. Section 2.3.3 contains examples of this latter interpretation. The event type, processor id, and process id fields may also have other special values that denote subsets. Some of these values are predefined, as in Table 5, but most would be defined by special record types, described in §2.3.1 and §2.3.8. These subset field values are used for indicating that the given information applies to the given subset, of processors, processes, or event types, and are not used for any other purposes, in contrast to the wildcard value.

2.3. PICL/ParaGraph-specific interpretation of field values

In this section, we first describe the new record and event types. We then describe the data associated with a given record type and event type.

2.3.1. Record types

The record types currently supported are described in Table 3. There are four record type categories: user-defined record types, event record types, statistics record types, and subset-

Record Type	*Meaning*
≥ 0	user-defined record type
-2	event mark
-3	event entry
-4	event exit
-5	event label
-6	event data descriptor
-7	event message
-101	data fields contain cumulative time spent in (other) event types, relative to specified event type
-102	data fields contain number of occurrences of (other) event types, relative to specified event type
-103	data fields contain the volume statistic for (other) event types, relative to specified event type
-201	data fields contain a list of processors defined as a subset
-202	data fields contain a list of processes defined as a subset
-203	data fields contain a list of event types defined as a subset

Table 3: PICL record types

definition record types. The user-defined record types are for identifying (to the user) what the associated data means for a given event. The event record types are for collecting the detailed event information, for both system and user-defined events, needed for an animation system like ParaGraph. The statistics record types are for sampling performance data and for profiling system and user-defined event types. For example, they can record the amount of time that a process spent waiting for messages to arrive while within user-defined events of a given type. The subset-definition record types are for defining values that refer only to a subset of processors, processes, or event types.

The event entry and exit records are used for "events" (or states) that have a measurable duration. The event mark is used for events that have no duration, or whose duration is too short to be measured reliably. Note that whether an event has a measurable duration or not can be a function of the instrumentation package, and it is legal to use either entry/exit or mark record types with any event.

The event label record is for "naming" an event type, and is expected to have character data. By use of wildcard or subset field values, an event name can be assigned that applies to all occurrences of an event type within a set of processes or processors. The event data descriptor record is for associating an integer with a control string (for use in subsequent records with this event type). This record has a single data field that contains an integer followed by a `scanf` control string. Again, wildcard and subset values can be used to "globally" associate an integer with a particular control string.

The event message record is for recording error and debugging messages, and is expected to have character data. While not part of the format specification, the event message record is

unique in PICL in that it is written to the trace file immediately instead being saved in local storage until the entire trace buffer is flushed. In consequence, this record is saved even when the program does not run to completion.

Currently, three statistics record types are defined. Each statistics record type is for collecting a specific type of information. For example, suppose that a user defines the execution of a subroutine to be an event of a certain type. Then a statistics record of type -101 would be used to describe how much time had been spent executing each PICL command during all calls to this subroutine. Similarly, a statistics record of type -102 would be used to describe how many times other user-defined event types had occurred within calls to the subroutine. The volume statistic collected by record type -103 is described in §2.3.7. Note that the best approach for collecting performance statistics is still under investigation, and that the number of statistics record types is likely to grow. But we expect no significant change in the definitions of the record types already specified.

2.3.2. Event types

The event types in the new PICL trace file format are listed in Tables 4 and 5. There are currently 9 major categories of events: user-defined events, all events (wildcard), interprocess communication events, file i/o events, synchronization events, resource allocation events, generic performance events (for labelling user code), tracing events, and predefined event subsets. Each of these categories is further subdivided, as indicated in the tables. The categories and numbering scheme have been chosen to make it simple to add event types.

Note that some event types are not associated with PICL commands. Some of these event types reflect extensions that we intend to make to PICL in the near future. Some of these event types are internal to PICL and are not associated with any one command. Finally, some of these event types are for referring to predefined subsets of event types. These subset types are primarily for use in statistics records. See §2.3.7 for an explanation of their use.

PICL does not produce event records for some of the event types in Tables 4 and 5, but these event types still have associated performance statistics. For example, PICL does not produce event records for **probe0** events (event types -53 and -54), but the number of times that each of these event types occurs is recorded. (Some programs make heavy use of **probe0**, e.g. event loops in which **probe0** is called to check whether a message has arrived that needs to be processed. Producing trace records for each **probe0** call in such a program would produce very large trace files and could significantly perturb the behavior of the program.)

Comments on some of the event categories follow.

Event type	*PICL command*	*Event*
≥ 0	—	user-defined events
-1	—	matches all events (wildcard)
-11	`open0`	enabling interprocess communication
-12	`close0`	disabling interprocess communication
-13	`who0`	requesting interprocess communication parameters
-14	`recvinfo0`	requesting information about most recently received or queried after message
-21	`send0`	sending a message (blocking)
-27	`sendbegin0`	posting a request that a message be sent (nonblocking)
-28	`sendstatus0`	checking and finding that a send request is complete
-29	`sendstatus0`	checking and finding that a send request is not complete
-30	`sendend0`	verifying that a send request is complete
-31	`sendend0`	waiting until a send request is complete
-51	`recv0`	receiving a message (blocking)
-52	`recv0`	waiting to receive a message (blocking)
-53	`probe0`	checking and finding that a message is available to be received
-54	`probe0`	checking and finding that no message is available to be received
-55	`wait0`	verifying that a message is available to be received
-56	`wait0`	waiting until a message is available to be received
-57	`recvbegin0`	posting a request that a message be received (nonblocking)
-58	`recvstatus0`	checking and finding that a receive request is complete
-59	`recvstatus0`	checking and finding that a receive request is not complete
-60	`recvend0`	verifying that a receive request is complete
-61	`recvend0`	waiting until a receive request is complete
-201	—	opening an i/o channel or initializing a file descriptor
-202	—	closing an i/o channel or freeing a file descriptor
-221	—	writing to a channel or file
-251	—	reading from a channel or file
-401	`clocksync0`	synchronizing process clocks
-402	`sync0`	synchronizing processes
-501	`open0`	allocating computing resources
-502	`close0`	releasing computing resources
-503	`load0`	spawning process(es)
-601	—	idle (generic)
-602	—	system overhead (generic)
-603	—	user or parallel overhead (generic)

Table 4: PICL event types

Event type	*PICL command*	*Event*
-901	`tracenode`	tracing
	`tracehost`	
	`traceexit`	
-902	`traceevents`	specifying number of user event types to collect statistics for
-903	`tracefiles`	specifying file in which to save the trace data
-904	`tracelevel`	specifying what trace data to generate
-904	`traceinfo`	requesting information on current tracing parameters
-911	`tracemsg`	placing a message in the trace file
-912	`traceflush`	writing the trace data to the trace file
-913	—	trace buffer has filled up, and detailed tracing has been disabled
-914	—	trace buffer has filled up, and trace records are being overwritten
-1001	—	matches all user events
-1002	—	matches all system events
-1011	—	matches all interprocessor communication events (types -11 – -200)
-1012	—	matches all send events (types -21 – -50)
-1013	—	matches all receive events (types -51 – -80)
-1014	—	matches all blocked send/receive events (types -31, -52, -56, -61)
-1201	—	matches all file i/o events (types -201 – -400)
-1401	—	matches all synchronization events (types -401 – -500)
-1501	—	matches all resource allocation performance events (types -501 – -600)
-1601	—	matches all generic performance events (types -601 – -700)
-1901	—	matches all tracing events (types -901 – -1000)
-1902	—	matches all trace flushing events (types -911, -912)

Table 5: PICL event types

Interprocess communication events. This category dominates in PICL trace files that are used as input to ParaGraph, and the numbering scheme has been chosen to keep the event type numbers small. The semantics for the send and receive events listed in Table 4 are not completely defined. When sending a message on iPSC and nCUBE multiprocessors, the message is copied into a system buffer if the destination process has not yet posted a corresponding receive request. On these systems, the send operation is complete (as far as the sending process is concerned) when the send buffer can be modified without corrupting the message. On a system using fully synchronous message-passing semantics, a send operation is not complete until the message has been received by the destination process. Both blocking (**send0** and **recv0**) and nonblocking (**sendbegin0**, **recvbegin0**) communication events are meaningful for both styles of message-passing semantics. In blocking events, control does not return to the calling process until the corresponding operation is complete. In nonblocking events, control returns immediately, and further events (commands) are required to determine when the corresponding operation is complete, e.g. **sendstatus0**, **sendend0**, **recvstatus0**, and **recvend0**. Thus, the difference between the two styles of message-passing semantics is solely in the definition of when a send operation is complete. If PICL, or some other system, ever supports both styles of message-passing semantics in the same program, then additional event types will need to be defined. This possibility was taken into account in the current numbering scheme.

Note that a call to **recv0** that waits because the message has not yet arrived and a call to **recv0** that does not need to wait are treated as separate event types. Systems that can differentiate between the "overhead" of processing a receive request and the time when a process is idle waiting for a message to arrive may wish to nest the two event types, to record this information. Since PICL cannot do this either one event type or the other will be associated with a given **recv0** call, but not both. Similar comments apply to the **sendend0**, **wait0**, and **recvend0** PICL commands (and corresponding event types).

File i/o events. File input/output events strongly influence the performance of parallel application codes. Rather than defining PICL calls for these commands, we decided to provide support for hand instrumentation of the relevant commands. This decision is likely to change when it becomes clear what type of file i/o commands should be supported directly in PICL, or when a standard emerges. In the meantime, we anticipate that the number of distinct file i/o event types will eventually be comparable to the number of interprocess communication event types, and have chosen the numbering scheme accordingly.

Missing events. Events that have been ignored include more sophisticated point-to-point communication events like swap, all global events other than simple synchronization, and any

support for shared-memory primitives (locks, semaphores, etc.) We have attempted to leave room in this numbering scheme for a significant increase in the number of event types. But renumbering the event types listed in Tables 4 and 5 will be a simple task if the current numbering system must eventually be redone.

2.3.3. Data fields for event entry, exit, and mark records

Tables 6–18 describe the data fields associated with event entry/exit or event mark record types for PICL events. As indicated earlier, event mark and event entry/exit are both legal record type "options" for any event. PICL currently produces either an event mark or an event entry/exit pair for any given event type, but not both, and only the records generated by PICL are described in Tables 6–13. But the event mark records are easily specified in terms of the event entry/exit records; the data fields for an event mark record are the data fields for the corresponding event entry record followed by the data fields for the event exit record. In contrast, specifying event entry/exit records from event mark record specifications must be done on a case-by-case basis. For the current set of PICL event types, it is sufficient to put all of the data fields from the event mark record in the event entry record.

For the most part, *we consider all data fields to be optional.* An animation tool needs some of the data for certain displays, like destination for a send command and source for a receive command when displaying a spacetime diagram [7], but other displays need only the time that an event began and ended to be meaningful. It is our hope that analysis programs will use the data, if available, but be robust enough to work if the data are missing.

Wildcard event. Entry and exit records for the wildcard event are one way of indicating the beginning and end (of the traced portion) of a process. Note that this is redundant because the tracing event records indicate exactly the same information. The wildcard is the natural event type for describing cumulative statistics for an entire process, but the tracing event type also has the same statistics.

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
-1	(all events)	entry	0	—	—
		exit	0	—	—

Table 6: PICL data fields for wildcard event

Interprocess communication events. The data fields of the interprocess communication events (Tables 7 and 8) reflect PICL's calling syntax and semantics, but also provide explicit support for multiple processes per processor selecting messages by processor and process id as

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
-11	`open0`	entry	0	—	—
		exit	0	—	—
-12	`close0`	mark	0	—	—
-13	`who0`	—	—	—	—
-14	`recvinfo0`	—	—	—	—
-21	`send0`	entry	4 (3)	"%d"	length in bytes type destination processor id destination process id
		exit	0	—	—
-27	`sendbegin0`	entry	4 (3)	"%d"	length in bytes type destination processor id destination process id
		exit	1	"%d"	send request id
-28	`sendstatus0`	entry	1	"%d"	send request id
		exit	0	—	—
-29	`sendstatus0`	—	—	—	—
-30	`sendend0`	entry	1	"%d"	send request id
		exit	0	—	—
-31	`sendend0`	entry	1	"%d"	send request id
		exit	0	—	—
-51	`recv0`	entry	3 (1)	"%d"	requested type requested processor id requested process id
		exit	4 (3)	"%d"	length in bytes received type source processor id source process id

Table 7: PICL data fields for interprocess communication events

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
-52	**recv0**	entry	3 (1)	"%d"	requested type requested processor id requested process id
		exit	4 (3)	"%d"	length in bytes received type source processor id source process id
-53	**probe0**	—	—	—	—
-54	**probe0**	—	—	—	—
-55	**wait0**	—	—	—	—
-56	**wait0**	entry	3 (1)	"%d"	requested type requested processor id requested process id
		exit	4 (3)	"%d"	length in bytes found type source processor id source process id
-57	**recvbegin0**	entry	3 (1)	"%d"	requested type requested processor id requested process id
		exit	1	"%d"	receive request id
-58	**recvstatus0**	entry	1	"%d"	receive request id
		exit	4 (3)	"%d"	length in bytes received type source processor id source process id
-59	**recvstatus0**	—	—	—	—
-60	**recvend0**	entry	1	"%d"	receive request id
		exit	4 (3)	"%d"	length in bytes received type source processor id source process id
-61	**recvend0**	entry	1	"%d"	receive request id
		exit	4 (3)	"%d"	length in bytes received type source processor id source process id

Table 8: PICL data fields for interprocess communication events

well as by message type, and typeless messages, none of which are currently supported in the PICL programming model. The wildcard value (-1) can be used to indicate that a given data field is not relevant or not defined. Data fields at the end of the record can also simply be dropped by stipulating a smaller value for the number of data fields. The ordering of the data fields has been chosen so that this latter mechanism can be used to include only information relevant to PICL. The number of data fields that will be produced in the next version of PICL is indicated in parentheses.

Simply eliminating data fields that are not relevant is not a very general technique for adding flexibility, depending as it does on both the choice and the ordering of the data fields as to whether the approach can be used. But it does decrease the size of trace records, and it does follow the previously mentioned maxim that missing data should not effect correct execution of visualization or analysis tools. In particular, both eliminating fields and using the wildcard field value should be (equally) valid techniques for indicating that a certain data field is not relevant or not defined. A different option for incorporating different versions of send and receive is to introduce new event types for these operations, but this should probably be done only for truly different events, e.g. send/receive with significantly different semantics.

As indicated in §2.3.2, PICL does not produce event records for event types -53 and -54, associated with **probe0**, because these events are of short duration and may occur very often in a program. For similar reasons, PICL does not produce event records for event types -13, -14, -29, -55, and -59, associated with **who0**, **recvinfo0**, **sendstatus0**, **wait0**, and **recvstatus0**, respectively. But PICL will produce event records for event types -28, -56, and -58, also associated with **sendstatus0**, **wait0**, and **recvstatus0**, either because of the longer duration of the events or because of the limited frequency that these events will occur. For example, for each nonblocking receive request, at most one call to **recvstatus0** will find that the message has been received (event type -58). Subsequent calls to **recvstatus0** will check for the completion of the next outstanding receive request of the indicated type, returning an error if there are no other outstanding receive requests of that type.

Note that, in PICL, the send request id for the **sendstatus0** and **sendend0** events is identical to the message type associated with the corresponding **sendbegin0** event. If instrumenting the native iPSC equivalents (**msgdone**, **msgwait**, and **isend**), then a different value would be used to identify the corresponding **isend**, but the definition of the data field applies to both cases. Similar comments apply to the receive request id.

File i/o events. File i/o events are one of the recent additions to the PICL tracing facility, and we are still working out how best to instrument them. For example, there are numerous different types of file read and write commands, each of which may require a distinct event type. Minimally, we want to know how often these events occur, and how much time is spent

within them. As indicated in Table 9, we also record the name of the file or channel that is opened or closed in file open or close commands, what file is being written to or read from in read or write commands, and the amount of data read or written.

Synchronization events. The only synchronization events specified currently are the clock normalization and barrier events. While PICL does not support barriers for subsets of processors or processes, this is an obvious (and important) generalization, and the data fields have been chosen to support this. The number of data fields actually generated by PICL is indicated in parentheses, as before, although the wildcard value is an equally valid way of indicating that all processes and processors are expected to participate in the event.

Note that many of the shared-memory primitives, like locks and semaphores, would also be useful additions to this category of event types. While this trace format is strongly geared toward message-passing systems, the new generation of multiprocessors support multiple programming models, and a single program may mix programming styles.

Resource allocation events. The only support in PICL for resource allocation is that the host processor (on systems with a host) can allocate and deallocate processors and load processes onto processors. Since the format described in Table 11 does not restrict these events to the host processor, the same events can be used for the next generation of multiprocessors or on networks of workstations, where any process can allocate processors and load (spawn) other processes. But for these more general resource allocation and process spawning events, more or different data may need to be recorded.

Generic performance events. Not all events of interest are defined within this format, and generic labels are provided that can be used to identify performance aspects of an unidentified event. For example, event type -603 can be used to label sections of code that are added when parallelizing a serial code, to support the identification of parallel inefficiencies or overhead. The entry/exit records associated with these events are described in Table 12.

Tracing events. Tracing events are recorded to allow the trace file to be interpreted correctly, and to measure some of the gross effects of tracing. See Table 13 for a description of the data fields associated with these events. Note that the first event record is always the `tracenode/tracehost` entry record and the last event record is always the corresponding exit record. Thus, cumulative statistics for an entire process are the relative statistics for the "tracing" event type (-901).

Subset events. Entry, exit, and mark records for subsets of event types are "legal", but not particularly useful or meaningful. The primary use of subset event types is in the specification

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
-201	(file open)	entry	(variable)	"%c"	file/channel name
		exit	1	"%d"	file descriptor/unit number
-202	(file close)	entry	1	"%d"	file descriptor/unit number
		exit	0	—	—
-221	(file write)	entry	2	"%d"	length in bytes file descriptor/unit number
		exit	0	—	—
-251	(file read)	entry	0	—	—
		exit	2	"%d"	length in bytes file descriptor/unit number

Table 9: PICL data fields for file i/o events

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
-401	`clocksync0`	entry	2 (0)	"%d"	processor subset synchronized process subset synchronized
		exit	0	—	—
-402	`sync0`	entry	2 (0)	"%d"	processor subset synchronized process subset synchronized
		exit	0	—	—

Table 10: PICL data fields for synchronization events

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
-501	`open0`	entry	1	"%d"	requested number of processors
		exit	1	"%d"	allocated number of processors
-502	`close0`	entry	0	—	—
		exit	0	—	—
-503	`load0`	entry	(variable)	"%c"	name of executable
		exit	2	"%d"	processor on which process was spawned process id of spawned process

Table 11: PICL data fields for resource allocation events

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
-601	(idle)	entry	0	—	—
		exit	0	—	—
-602	(system overhead)	entry	0	—	—
		exit	0	—	—
-603	(user-defined or parallel overhead)	entry	0	—	—
		exit	0	—	—

Table 12: PICL data fields for generic performance events

of cumulative statistics for the subsets, especially in the context of statistics records. The number of predefined subset event types is likely to increase in the future, as new (individual) event types are added, and as new subsets are identified as being important.

2.3.4. Data fields for event label records

The event label record is for "naming" an event type occuring in a given process and processor. By use of wildcard (or subset) values, this record type can also be used to label all instances of an event type, independent of the process and processor, or to label a process or processor independent of the event type. As indicated in Table 14, the data should be a sequence of characters defining the label.

2.3.5. Data fields for event data descriptor records

The event data descriptor record is for associating an integer with a control string. As indicated in Table 15, this record has a single data field that contains an integer followed by a character string, which is assumed to be a double-quote delimited `scanf` control string. The scope of the integer alias for the control string can be controlled by the use of wildcard or subset field values. Note that `"%d%s"`, the data descriptor for the trace record indicated in Table 15, will not adequately describe the format of the data field if the control string within the field contains white space characters. In such a case, the data descriptor should be modified.

2.3.6. Data fields for event message records

The event message record is for recording error and debugging messages, and other comments or annotations that are needed in the trace file. As indicated in Table 16, the data should be a sequence of characters defining the label.

2.3.7. Data fields for statistics records

Each event type has cumulative statistics representing time spent in events of this type and the number of times events of this type occurred. Each event type with a length data field (types associated with `send0`, `sendbegin0`, `recv0`, `wait0`, `recvstatus0`, `recvend0`, file write, file read, `tracemsg`, and `traceflush`) has a corresponding cumulative volume statistic. The number of processes spawned by `load0` is also considered to be a volume statistic. Finally, the total amount of trace data collected by a process is considered to be a volume statistic for the tracing event type (-901).

If all events of a given type have durations that are not measurable or are not measured, then the associated time statistic value is -1. Similarly, if the number of occurrences or the volume for an event type is not measured or is not defined, then the corresponding statistic

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
-901	`tracenode/ tracehost`	entry	0	—	—
	`traceexit`	exit	0	—	—
-902	`traceevents`	entry	1	"%d"	number of user events being profiled
		exit	0	—	—
-903	`tracefiles`	entry	0	—	—
		exit	0	—	—
-904	`tracelevel`	mark	3	"%d"	PICL event tracing level user event tracing level trace event tracing level
-905	`traceinfo`	–	—	—	—
-911	`tracemsg`	entry	1	"%d"	length in bytes
		exit	0	—	—
-912	`traceflush`	entry	0	—	—
		exit	1	"%d"	length in bytes
-913	(buffer full)	mark	0	—	—
-914	(overwriting buffer)	mark	0	—	—

Table 13: PICL data fields for tracing events

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
(any)	—	-5	(variable)	"%c"	label

Table 14: PICL data fields for event label records

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
(any)	—	-6	1	"%d%s"	control string

Table 15: PICL data fields for event data descriptor records

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
(any)	—	-7	(variable)	"%c"	message

Table 16: PICL data fields for event message records

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
(any)	—	-101	(variable)	"%d%lf"	event type and cumulative time spent within event type
(any)	—	-102	(variable)	"%d%d"	event type and number of occurrences of event type
(any)	—	-103	(variable)	"%d%d"	event type and associated volume statistic

Table 17: PICL data fields for statistics records

value is -1. Thus, an instrumentation package does not have to collect all of the data specified in the new PICL trace file format. In particular, PICL may not collect all of these statistics.

The data fields in the statistics records are described in Table 17. The number of data fields is variable, but each field is an ordered pair consisting of an event type and an associated statistic. The statistic for the associated event type is measured relative to the event type of the trace record. As mentioned above, for event types without a particular statistic, the corresponding data field is defined to be -1. Similarly, if an event type has not been defined, then the associated statistics are defined to be -1. To keep the statistics records short, PICL only outputs these ordered pairs for events with postive statistics, with the underlying assumption that those events not mentioned have zero or undefined statistics.

A complete profile of PICL and/or user event types is useful as a summary statistic, but it is both very space consuming and expensive to calculate if sampled multiple times during a code's execution. A more practical approach is to specify subsets of these event types (e.g. all sends, all receives, etc.), and return statistics summed over subsets. Some standard subsets have been defined in Table 5 for this purpose, and other subsets can be defined using the subset-definition record types.

2.3.8. Data fields for subset-definition records

The data fields in the subset-definition records are described in Table 18. The number of data fields is variable, but the first field defines the integer label for the subset, and the subsequent fields define the subset.

Event type	*PICL command*	*Record type*	*# of Data Fields*	*Data Descriptor*	*Data Fields*
(any)	—	-201	(variable)	"%d"	subset identifier and list of processor ids in subset
(any)	—	-202	(variable)	"%d"	subset identifier and list of process ids in subset
(any)	—	-203	(variable)	"%d"	subset identifier and list of event types in subset

Table 18: PICL data fields for subset-definition records

2.4. Header records

Header records are trace records that are prepended to a trace file and that are used in the processing or understanding of the rest of the trace file. The one example mentioned previously is the use of header records (in some standard metaformat) to define the format of subsequent records. A more typical use of header records is to identify the trace file: when it was generated, who generated it, the program(s) that produced the trace, the parallel computer the program

ran on, the configuration of the parallel computer, etc. Also, some visualization and analysis tools expect the trace file to begin with global information on the number of processes, the number of events, event labels, etc., so that the tool does not have to read the trace file more than once. Other information, relevant to PICL-like systems, that can be encoded in header records include clock resolution(s), integer aliases for data descriptor control strings, integer aliases for files names and i/o channels, processor and process subset definitions, names of executables associated with processor and/or process ids, machines names corresponding to processor ids, and lines of source code and names of source files corresponding to events. (The last item is of major importance in debugging, and the second to last item is important when computing over heterogeneous networks, where the processor numbering is arbitrary, but the machine names are fixed.) Some of this information is, or can be, embedded within the trace file, but the visualization tools want the information at the beginnining of the file.

There are as yet no PICL trace records defined specifically as header records. Producing header records is problematic in PICL because there is no controlling process in the writing of a trace file, and PICL header records will need to be added in a postprocessing step. But the format of header records is already fairly well determined. Some new record types may be needed (probably with type values > 200), but wildcard or subset field values can be used to indicate the scope of the the information. The timestamp for header records added in postprocessing should precede the earliest recorded time, so that they are still the first records in the file after sorting by timestamp. Due to the clock normalization logic, negative timestamps are legal, so it is not sufficient simply to define the timestamp of a header record to be less than zero.

2.5. ParaGraph requirements

PICL always produces entry and exit records for the tracing event (-901) and detailed summary statistics (record types -101, -102, -103) for event type -1. ParaGraph looks for the tracing event entry record to begin the animation for a given process, and uses the tracing event exit record to determine when the process has terminated. ParaGraph then uses the mark, entry, and exit record types for other events, both system and user-defined, when animating the trace file. User-defined record types are sometimes used in special application specific ParaGraph displays, where the record types define what the associated data means. Currently, ParaGraph ignores the event label, data desciptor, and message records, all statistics records, and all subset-definition records.

ParaGraph expects that there will be exactly one process per processor, and that P processors are numbered 0 to $P-1$, with the exception of a host processor. In homogeneous multiprocessor systems with a host, the host processor generally behaves much differently than

the other processors, and it is important to distinguish it in the trace data. In PICL, the host processor id is 32767, and ParaGraph ignores any trace records with this processor id.

3. Future work

The new PICL trace file format is not completely specified. But the part described in this document already represents a superset of the old PICL format, and it is sufficiently well-defined to be used with ParaGraph. Further development will be driven by our experiences with the new format, especially in the context of our tuning and modeling research. We also welcome suggestions and comments from users and from developers of similar systems.

Like PICL, the new PICL trace file format is a research tool, and the needs of our research will determine how it changes. But we anticipate most future changes to be restricted to the PICL/ParaGraph-specifics described in §2.3, and those changes are most likely to be the addition of new event and record types and the addition of new data fields to existing record types.

Acknowledgements

The design of the new PICL trace file format was motivated primarily by the needs of ORNL projects in performance modeling and in global climate change, but the following people also made contributions. Extensive discussions with Maurice van Riek, then at LIP in Lyon, convinced us of the utility of making the new format general enough to be used in other systems. Discussions with Michael Heath and our experience with a prototype version of ParaGraph written by Jennifer Finger confirmed that the new format works well with ParaGraph. We express our appreciation to Jon Flower of Parasoft for providing us with descriptions of the trace data formats used in Express. We also express our appreciation to Ewing Lusk of Argonne National Laboratory for providing descriptions of the trace data formats produced by the `alog` package and used in the visualization tool `upshot`, and for commenting on the appropriateness of the new PICL format for these systems.

A. Example trace file

The trace records in Table 20 were generated by the PICL program in Table 19 running on 8 processors of an Intel iPSC/860 at Oak Ridge National Laboratory. The program broadcasts the time from processor 0 to the other processors twice, once by using the default broadcast mechanism (a destination value of -1) and once by using a ring broadcast algorithm. Tracing commands are used to identify the first broadcast as user event 0 and the second broadcast as user event 1. The value received in each broadcast is saved in the trace file using a user-defined

record type. To keep the trace file short, the program specifies that trace data from only one processor be saved in the file **trace**. Since this program was run "hostless", i.e. without a host program, the trace data generated on the other processors was lost.

To demonstrate the relevant trace records, all PICL commands are called after tracing is enabled. In practice, initial calls to **tracelevel**, **traceevents**, and **tracefiles** would be made *before* calling **tracenode**, and the clock synchronization option in **tracenode** would be used to mask the effect of the overhead introduced by these commands on the runtime behavior. In this program, the synchronization option is not invoked in the call to **tracenode** and an explicit call to **clocksync0** is made following the **tracefiles** call. This also masks the effect of the perturbation, but has the side-effect of making times preceding the call to **clocksync0** negative, which *is* legal in the new PICL format. (The timestamps of all trace records that have not yet been written to the permanent trace file are effected by synchronization of the clocks, not just those that follow the call to **clocksync0**.) If the call to **clocksync0** were replaced with a call to **tracenode** with the clock synchronization option set, then the positive timestamps would be similar, and the trace records with negative timestamps would not have been generated.

The program uses some PICL commands that are not yet available in the released version of the library, but the comments in the program should be sufficient to interpret the code. Each line of code has been numbered, and numbers have been added to the trace in Table 20 to indicate which line of code caused the trace record to be generated. These line numbers do not actually occur in the program or trace file. Also, trace records that are too long to be displayed on one line have the symbol "\" appended to the end of the line that is broken.

Note that many of the trace records are associated with the **close0** call. Since **traceexit** is not called explicitly, **close0** calls it implicitly, at which time all of the summary statistics are generated. As an aid in interpreting Table 20, Table 21 describes the first statistics record in detail. As mentioned in §2.3.7, only those events whose statistics are positive are listed in the statistics records.

```
(1)  main()
(2)  {
(3)    int numproc,me,host;
(4)    double x,clock0();
(5)
(6)    tracenode(10000,0,0);        /* enable tracing */
(7)    tracelevel(1,1,1);           /* set tracing levels */
(8)    traceevents(2,1,1,1,1,1);    /* enable profiling of user events */
(9)    open0(&numproc,&me,&host);   /* enable interprocessor communication */
(10)   if (me == numproc-2)
(11)     tracefiles("","trace",0);  /* specify where to save trace data */
(12)
(13)   clocksync0();                /* synchronize processor clocks */
(14)   traceblockbegin(0,0,0);      /* record beginning of user event */
(15)   if (me == 0){                /* broadcast time using native broadcast */
(16)     x = clock0();
(17)     send0(&x,sizeof(double),0,-1);
(18)     }
(19)     else recv0(&x,sizeof(double),0);
(20)   traceblockend(0,0,0);        /* record end of user event */
(21)   tracedata(0,0,"double",1,&x);/* record data associated with user event */
(22)
(23)   traceblockbegin(1,0,0);      /* record beginning of user event */
(24)   if (me == 0){                /* broadcast time using ring algorithm */
(25)     x = clock0();
(26)     send0(&x,sizeof(double),1,me+1);
(27)     }
(28)     else{
(29)     recv0(&x,sizeof(double),1);
(30)     if (me+1<numproc)
(31)       send0(&x,sizeof(double),1,me+1);
(32)     };
(33)   traceblockend(1,0,0);        /* record end of user event */
(34)   tracedata(1,0,"double",1,&x);/* record data associated with user event */
(35)
(36)   close0();                    /* disable interprocessor communication */
(37) }
```

Table 19: Example PICL program

```
-3 -901 -0.715036 6 0 0                                                    (6)
-2 -904 -0.715024 6 0 3 2 1 1 1                                            (7)
-3 -902 -0.715017 6 0 1 2 2                                                (8)
-4 -902 -0.713847 6 0 0                                                    (8)
-3 -11 -0.713833 6 0 0                                                     (9)
-4 -11 -0.713735 6 0 0                                                     (9)
-3 -903 -0.713724 6 0 0                                                    (11)
-4 -903 -0.008091 6 0 0                                                    (11)
-3 -401 -0.008079 6 0 0                                                    (13)
-4 -401 0.000005 6 0 0                                                     (13)
-3 0 0.000016 6 0 2 2 0 0                                                  (14)
-3 -52 0.000128 6 0 1 2 0                                                  (19)
-4 -52 0.000516 6 0 3 2 8 0 0                                              (19)
-4 0 0.000539 6 0 2 2 0 0                                                  (20)
0 0 0.000635 6 0 1 5 0.000124                                              (21)
-3 1 0.000711 6 0 2 2 0 0                                                  (23)
-3 -52 0.000818 6 0 1 2 1                                                  (29)
-4 -52 0.001643 6 0 3 2 8 1 5                                              (29)
-3 -21 0.001665 6 0 3 2 8 1 7                                              (31)
-4 -21 0.001711 6 0 0                                                      (31)
-4 1 0.001724 6 0 2 2 0 0                                                  (33)
0 1 0.001971 6 0 1 5 0.000410                                              (34)
-2 -12 0.001979 6 0 0                                                      (36)
-4 -901 0.001982 6 0 0                                                     (36)
-101 -1 0.001982 6 0 7 "%d%lf" -11 0.000098 -21 0.000046 -52 0.001212 \    (36)
 -401 0.008083 -901 0.717018 -902 0.001170 -903 0.705632
-102 -1 0.001982 6 0 9 "%d%d" -11 1 -12 1 -21 1 -52 2 -401 1 -901 1 \      (36)
 -902 1 -903 1 -904 1
-103 -1 0.001982 6 0 3 "%d%d" -21 8 -52 16 -901 428                        (36)
-101 -1 0.001982 6 0 2 "%d%lf" 0 0.000523 1 0.001013                       (36)
-102 -1 0.001982 6 0 2 "%d%d" 0 1 1 1                                      (36)
-101 0 0.001982 6 0 1 "%d%lf" -52 0.000387                                 (36)
-102 0 0.001982 6 0 1 "%d%d" -52 1                                         (36)
-103 0 0.001982 6 0 1 "%d%d" -52 8                                         (36)
-101 1 0.001982 6 0 2 "%d%lf" -21 0.000046 -52 0.000825                    (36)
-102 1 0.001982 6 0 2 "%d%d" -21 1 -52 1                                   (36)
-103 1 0.001982 6 0 2 "%d%d" -21 8 -52 8                                   (36)
```

Table 20: Trace records produced by example PICL program

Field name	*Field value*		*Meaning*
record type	-101		a statistics record containing cumulative event times
event type	-1		statistics are relative to the wildcard event, thus are summary statistics for the entire traced portion of the program
timestamp	0.001982		statistics valid at time .001982
processor id	6		statistics valid for processor 6
process id	0		statistics valid for process 0
number of data fields	7		seven cumulative time statistics recorded
data descriptor	"%d%lf"		each statistics data field is made up of an integer identifying the event and a double precision floating point time value
data field	-11	0.000098	time spent in `open0`
data field	-21	0.000046	time spent in `send0`
data field	-52	0.001212	time spent blocked waiting for a message in `recv0`
data field	-401	0.008083	time spent in `clocksync0`
data field	-901	0.717018	time spent between `tracenode` and `traceexit` (called implicitly inside `close0`)
data field	-902	0.001170	time spent in `traceevents`
data field	-903	0.705632	time spent in `tracefiles`

Table 21: Example PICL trace record

B. References

[1] R. A. AYDT, *The Pablo self-defining data format*, Tech. Report, Department of Computer Science, University of Illinois at Urbana-Champaign, March 1992.

[2] D. BERGMARK, *Supercomputing '90 BOF session on standardizing parallel trace formats.* unpublished document available via anonymous ftp from `eagle.cnsf.cornell.edu` as bof.ps for the Postscript version and bof.tex for the LaTeXversion, November 1990.

[3] T. H. DUNIGAN, *A message-passing multiprocessor simulator*, Tech. Rept. ORNL/TM-9966, Oak Ridge National Laboratory, Oak Ridge, TN, 1986.

[4] G. A. GEIST, M. T. HEATH, B. W. PEYTON, AND P. H. WORLEY, *A machine-independent communication library*, in The Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications, J. L. Gustafson, ed., Golden Gate Enterprises, P.O. Box 428, Los Altos, CA, 1990, pp. 565–568.

[5] ——, *PICL: a portable instrumented communication library, C reference manual*, Tech. Report ORNL/TM-11130, Oak Ridge National Laboratory, Oak Ridge, TN, July 1990.

[6] ——, *A users' guide to PICL: a portable instrumented communication library*, Tech. Report ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, TN, August 1990.

[7] M. T. HEATH AND J. A. ETHERIDGE, *Visualizing performance of parallel programs*, Tech. Report ORNL/TM-11813, Oak Ridge National Laboratory, Oak Ridge, TN, May 1991.

[8] ——, *Visualizing the performance of parallel programs*, IEEE Software, 8 (1991), pp. 29–39.

[9] V. HERRARTE AND E. LUSK, *Studying parallel program behavior with upshot*, Tech. Report ANL/TM-91/15, Argonne National Laboratory, Argonne, IL, August 1991.

[10] R. W. HON, *A simple trace interchange format*, Tech. Report, Apple Computer, Inc., Cupertino, CA, May 1989.

[11] B. W. KERNIGHAN AND D. M. RITCHIE, *The C Programming Language*, Prentice-Hall, Englewood Cliffs, NJ, second ed., 1988.

[12] NCUBE CORPORATION, *NCUBE Users Handbook*, Foster City, CA, October 1987.

[13] D. A. REED, R. D. OLSEN, R. A. AYDT, T. M. MADHYASTHA, T. BIRKETT, D. W. JENSEN, B. A. A. NAZIEF, AND B. K. TOTTY, *Scalable performance environments for parallel systems*, in The Sixth Distributed Memory Computing Conference Proceedings, Q. F. Stout and M. Wolfe, eds., IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 562–569.

[14] M. SIMMONS, R. KOSKELA, AND I.BUCHER, eds., *Parallel Computer Systems: Performance Instrumentation and Visualization*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

[15] M. VAN RIEK AND B. TOURANCHEAU, *A general approach to the monitoring of distributed memory machines: A survey*, Research Report no. 91-28, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, Lyon, France, September 1991.

[16] P. H. WORLEY AND M. T. HEATH, *Performance characterization research at Oak Ridge National Laboratory*, in Parallel Processing for Scientific Computing, J. Dongarra, P. Messina, D. C. Sorenson, and R. G. Voigt, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 431–436.

INTERNAL DISTRIBUTION

1. B. R. Appleton
2–3. T. S. Darland
4. J. J. Dongarra
5. T. H. Dunigan
6. G. A. Geist
7. M. R. Leuze
8. C. E. Oliver
9. B. W. Peyton
10–14. S. A. Raby
15. C. H. Romine
16. T. H. Rowan
17–21. R. F. Sincovec
22. D. W. Walker
23–27. R. C. Ward
28–32. P. H. Worley
33. Central Research Library
34. ORNL Patent Office
35. K-25 Applied Technology Library
36. Y-12 Technical Library
37. Laboratory Records - RC
38–39. Laboratory Records Department

EXTERNAL DISTRIBUTION

40. Loyce M. Adams, Applied Mathematics, FS-20, University of Washington, Seattle, WA 98195

41. Donald M. Austin, 6196 EECS Building, University of Minnesota, 200 Union Street, S.E., Minneapolis, MN 55455

42. Loretta Auvil, Rome Laboratory/C3CB, 525 Brooks Road, Griffiss AFB, NY 13441-4505

43. Robert G. Babb, Oregon Graduate Center, CSE Department, 19600 N.W. Walker Road, Beaverton, OR 97006

44. Henri E. Bal, Vrije Universiteit, Department of Mathematics and Computer Science, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

45. Edward H. Barsis, Computer Science and Mathematics, P. O. Box 5800, Sandia National Laboratory, Albuquerque, NM 87185

46. Adam Beguelin, Carnegie Mellon University, School of Computer Science, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890

47. Robert E. Benner, Parallel Processing Division 1413, Sandia National Laboratories, P. O. Box 5800, Albuquerque, NM 87185

48. Philippe Berger, Institut National Polytechnique, ENSEEIHT, 2 rue Charles Camichel-F, 31071 Toulouse Cedex, France

49. Donna Bergmark, 745 E & TC Building, Hoy Road, Cornell University, Ithaca, NY 14853

50. Roger Berry, NCUBE Corporation, 4313 Prince Road, Rockville, MD 20853

51. Scott Berryman, Yale University, Computer Science Department, 51 Prospect Street, New Haven, CT 06520

52. Ansgar Boehm, Department of Computer Science III, University of Erlangen/ Nueremberg, Martensstrasse 3, D-8520 Erlangen, Germany

53. Roger W. Brockett, Harvard University, Pierce Hall, 29 Oxford Street Cambridge, MA 02138

54. James C. Browne, Department of Computer Sciences, University of Texas, Austin, TX 78712

55. Greg Burns, Trollius Project Leader, Academic Computing, The Ohio State University, 1224 Kinnear Rd., Columbus, OH 43212

56. Bill L. Buzbee, Scientific Computing Division, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307

57. Maria Calzarossa, Dipartimento di Informatica e Sistemistica, Università Degli Studi di Pavia, Via Abbiategrasso 209, I-27100 Pavia, Italy

58. Brian M. Carlson, Computer Systems Research Institute, University of Toronto, Toronto, Ontario M5S 1A1, Canada

59. Ron Casselman, Systems and Computer Engineering, Carleton University, Ottawa, Ontario K1S 5B6, Canada

60. Tony Chan, Department of Mathematics, University of California, Los Angeles, 405 Hilgard Avenue, Los Angeles, CA 90024

61. Jagdish Chandra, Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709

62. Doreen Y. Cheng, Principal Engineer, Computer Science Corporation, NASA Ames Research Center, MS 258-6, Moffett Field, CA 94035

63. Sung-Chi Chu, Computer Science Department, Radford University, Radford, Virginia

64. Melvyn Ciment, National Science Foundation, 1800 G Street N.W., Washington, DC 20550

65. Lyndon J. Clarke, Edinburgh Parallel Computing Centre, James Clerk Maxwell Building, The King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ, United Kingdom

66. Dennis Cottel, Naval Command, Control and Ocean Surveillance Center, RDT&E Division, San Diego, CA 92152-5000

67. Alva Couch, Department of Computer Science, Tufts University, Medford, MA 02155

68. Tom Crockett, ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665-5225

69. George Cybenko, Center for Supercomputing Research and Development, University of Illinois, 104 South Wright Street, Urbana, IL 61801-2932

70. Ron Daniel Jr., Cambridge University Engineering Department, Trumpington Street, Cambridge CB2 1PZ, United Kingdom

71. Helen Davis, Computer Science Department, Stanford University, Stanford, CA 94305

72. Michel Dayde, Institut National Polytechnique, ENSEEIHT, 2 rue Charles Camichel-F, 31071 Toulouse Cedex, France

73. John J. Dorning, Department of Nuclear Engineering Physics, University of Virginia Reactor Facility, Charlottesville, VA 22901

74. Craig Douglas, IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598-0218

75. Larry Dowdy, Computer Science Department, Vanderbilt University, Nashville, TN 37235

76. Iain S. Duff, Atlas Centre, Rutherford Appleton Laboratory, Chilton, Oxon OX11 0QX, England

77. Dannie Durand, IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

78. Derek Eager, Department of Computer Science and Engineering, Sieg Hall, FR-35, University of Washington, Seattle, WA 98195

79. Jean-Marc Fellous, Center for Neural Engineering, University of Southern California, Los Angeles, CA

80. Edward Felten, Department of Computer Science, University of Washington, Seattle, WA 98195

81. Charles Fineman, Ames Research Center, Mail Stop 269/3, Moffet Field, CA 94035

82. Jon Flower, Parasoft Corporation, 2500 E. Foothill Boulevard, Suite 205, Pasadena, CA 91107

83. Anthony Ford, Meiko Limited, 650 Aztec West, Bristol BS12 4SD, United Kingdom

84. Geoffrey C. Fox, NPAC, 111 College Place, Syracuse University, Syracuse, NY 13244-4100

85. Joan M. Francioni, Computer Science Department, University of Southwestern Louisiana, Lafayette, LA 70504

86. Offir Frieder, George Mason University, Science and Technology Building, Computer Science Department, 4400 University Drive, Fairfax, Va 22030-4444

87. Robert E. Funderlic, Department of Computer Science, North Carolina State University, Raleigh, NC 27650

88. Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47401

89. Alan George, Vice President, Academic and Provost, Needles Hall, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

90. Ian Glendinning, Department of Electronics and Computer Science, University of Southampton, Southampton, SO9 5NH United Kingdom

91. Gene Golub, Computer Science Department, Stanford University, Stanford, CA 94305

92. Andy Grant, Computer Graphics Unit, Manchester Computing Centre, University of Manchester, Oxford Rd, Manchester M13 9PL, United Kingdom

93. William D. Gropp, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

94. Eric Grosse, AT&T Bell Labs 2T-504, Murray Hill, NJ 07974

95. John L. Gustafson, Ames Laboratory, 236 Wilhelm Hall, Iowa State University, Ames, IA 50011-3020

96. Robert M. Haralick, Department of Electrical Engineering, Director, Intelligent Systems Lab, University of Washington, 402 Electrical Engineering Building, FT-10, Seattle, WA 98195

97. Leslie Hart, R/E/FS5, 325 Broadway, Boulder, CO 80303

98. Ann H. Hayes, Computing and Communications Division, Los Alamos National Laboratory, Los Alamos, NM 87545

99. Michael T. Heath, National Center for Supercomputing Applications, 4157 Beckman Institute University of Illinois, 405 North Mathews Avenue, Urbana, IL 61801-2300

100. Don E. Heller, Physics and Computer Science Department, Shell Development Co., P. O. Box 481, Houston, TX 77001

101. Rolf Hempel, GMD, Schloss Birlinghoven, Postfach 13 16, D-W-5205 Sankt Augustin 1, Germany

102. John L. Hennessy, CIS 208, Stanford University, Stanford, CA 94305

103. Charles J. Holland, Air Force Office of Scientific Research, Building 410, Bolling Air Force Base, Washington, DC 20332

104. Leah H. Jamieson, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907

105. Gary Johnson, Scientific Computing Staff, ER-7, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585

106. Lennart Johnsson, Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142-1214

107. Harry Jordan, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO 80309

108. Bo Kagstrom, Institute of Information Processing, University of Umea, 5-901 87 Umea, Sweden

109. Malvyn Kalos, Cornell Theory Center, Engineering and Theory Center Building, Cornell University, Ithaca, NY 14853-3901

110. Alan H. Karp, HP Labs 3U-7, Hewlett-Packard Company, 1501 Page Mill Road, Palo Alto, CA 94304

111. Kenneth Kennedy, Department of Computer Science, Rice University, P.O. Box 1892, Houston, TX 77001

112. Alan E. Klietz, Minnesota Supercomputer Center, Inc., Army High Performance Computing Research Center, University of Minnesota, Minneapolis, MN 55455

113. Charles Koelbel, Rice University, CITI/CRPC, P. O. Box 1892, Houston, TX 77251

114. Peter A. Krauss, Lehrstuhl fuer Rechnergestuetztes Entwerfen, Technische Universitaet, Muenchen, Germany

115. Domenico Laforenza, Parallel Processing Group, CNUCE - Istituto del CNA, Via Santa Maria 36, 56100 Pisa, Italy

116. Michael Langston, Department of Computer Science University of Tennessee, Knoxville, TN 37996-1301

117. Richard Lau, Office of Naval Research, 1030 E. Green Street, Pasadena, CA 91101

118. Robert L. Launer, Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709

119. E. D. Lazowska, Department of Computer Science and Engineering, Sieg Hall, FR-35, University of Washington, Seattle, WA 98195

120. James E. Leiss, Rt. 2, Box 142C, Broadway, VA 22815

121. Eric Leu, Swiss Federal Institute of Technology, Department of Computer Science, Operating System Laboratory, IN - Ecublens, CH-1015 Lausanne, Switzerland

122. Ted Lewis, Research Director, Oregon Advanced Computing Inst., 19500 NW Gibbs Boulevard #101, Beaverton, OR. 97006

123. Heather M. Liddell, Center for Parallel Computing, Department of Computer Science and Statistics, Queen Mary College, University of London, Mile End Road, London E1 4NS, England

124. Rik Littlefield, Pacific Northwest Laboratory, MS K1-87, P.O.Box 999, Richland, WA 99352

125. Ivo de Lotto, Dipartimento di Informatica e Sistemistica, Università Degli Studi di Pavia, Via Abbiategrasso 209, I-27100 Pavia, Italy

126. Ewing Lusk, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, MCS 221 Argonne, IL 60439-4844

127. Yves Maheo, IRISA, Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France

128. Allen D. Malony, Department of Computer and Information Science, University of Oregon, Eugene, OR 97403

129. James McGraw, Lawrence Livermore National Laboratory, L-306, P. O. Box 808, Livermore, CA 94550

130. Paul C. Messina, Mail Code 158-79, California Institute of Technology, 1201 E. California Boulevard, Pasadena, CA 91125

131. John Michalakes, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

132. Neville Moray, Department of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green Street, Urbana, IL 61801

133. Richard Muntz, Computer Science Department, University of California at Los Angeles, Los Angeles, CA 90024

134. David Nelson, Director, Scientific Computing Staff, ER-7, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585

135. Randolph Nelson, IBM, P.O. Box 704, Room H2-D26, Yorktown Heights, NY 10598

136. Joseph Oliger, Computer Science Department, Stanford University, Stanford, CA 94305

137. James M. Ortega, Department of Applied Mathematics, Thornton Hall, University of Virginia, Charlottesville, VA 22901

138. Steve Otto, Oregon Graduate Institute, Department of Computer Science and Engineering, 19600 NW von Neumann Drive, Beaverton, OR 97006-1999

139. Daniel Palermo, University of Illinois, 259 C&SRL, MC 228, 1308 West Main Street, Urbana, IL 61801

140. Cherri Pancake, Department of Computer Science, Oregon State University, Corvallis, OR 97331-3202

141. Merrell Patrick, Department of Computer Science, Duke University, Durham, NC 27706

142. David A. Poplawski, Department of Computer Science, Michigan Technological University, Houghton, MI 49931

143. Angela Quealy, Sverdrup Technology, Inc., 2001 Aerospace Parkway Brook Park, OH 44142

144. Daniel A. Reed, Computer Science Department, University of Illinois, Urbana, IL 61801

145. John R. Rice, Computer Science Department, Purdue University, West Lafayette, IN 47907

146. Maurice van Riek, Alex Informatique, 428, Place Jacques Cartier, Montreal, Quebec H2Y3B3, Canada

147. Bernhard Ries, Intel Corporation, European Supercomputer Development Center, Dornacher Str. 1, W-8016 Feldkirchen b. M., Germany

148. Diane T. Rover, 155 Engineering Building, Department of Electrical Engineering, Michigan State University, East Lansing MI 48824

149. Ahmed H. Sameh, Department of Computer Science, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455

150. Joel Saltz, ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665-522

151. Lionel Saugrain, Archipel S.A., P.A.E. des Glaisins, 1, rue du Bulloz, 74940 Annecy-Le-Vieux, France

152. Robert B. Schnabel, Department of Computer Science, University of Colorado at Boulder ECOT 7-7 Engineering Center, Campus Box 430, Boulder, CO 80309-0430

153. Martin H. Schultz, Department of Computer Science, Yale University, P. O. Box 2158 Yale Station, New Haven, CT 06520

154. James L. Schwarzmeier, Cray Research, Inc., 900 Lowater Road, Chippewa Falls, WI 54729

155. David S. Scott, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006

156. The Secretary, Department of Computer Science and Statistics, The University of Rhode Island, Kingston, RI 02881

157. Charles L. Seitz, Department of Computer Science, California Institute of Technology, Pasadena, CA 91125

158. Giuseppe Serazzi, Dipartimento di Scienze della Informazione, Università di Milano, Via Moretto da Brescia 9, I20133 Milano, Italy

159. Kenneth C. Sevcik, Computer Systems Research Institute, 10 King's College Road, University of Toronto, Toronto, Ontario M5S 1A1, Canada

160. Margaret L. Simmons, Computing and Communications Division, Los Alamos National Laboratory, Los Alamos, NM 87545

161. Horst D. Simon, NASA Ames Research Center, Mail Stop T045-1, Moffett Field, CA 94035

162. Tony Skjellum, Lawrence Livermore National Laboratory, L-316, P. O. Box 808, Livermore, CA 94550

163. Burton Smith, Tera Computer Company, 400 North 34th Street, Suite 300, Seattle, WA 98103

164. Wayne D. Smith, Intel Corporation, Supercomputer Systems Division, 15201 NW Greenbrier Parkway, Beaverton, OR 97006

165. Marc Snir, IBM T.J. Watson Research Center, Department 420/36-241, P. O. Box 218, Yorktown Heights, NY 10598

166. Larry Snyder, Department of Computer Science and Engineering, FR-35, University of Washington, Seattle, WA 98195

167. Danny C. Sorensen, Department of Mathematical Sciences, Rice University, P. O. Box 1892, Houston, TX 77251

168. Rick Stevens, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

169. G. W. Stewart, Computer Science Department, University of Maryland, College Park, MD 20742

170. Steven Suhr, Computer Science Department, Stanford University, Stanford, CA 94305

171. Paul N. Swarztrauber, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307

172. Bernard Tourancheau, LIP, ENS-Lyon, 69364 Lyon Cedex 07, France

173. Anne Trefethen, Engineering & Theory Center, Cornell University, Ithaca, NY 14853

174. Cecile Tron, IMAG, INPG (LMC), 46, av Felix Viallet, 38031 Grenoble Cedex, France

175. Sue Utter Honig, Computer Science Department, 4105C Upson Hall, Cornell University, Ithaca, NY 14853

176. Robert van de Geijn, University of Texas, Department of Computer Sciences , TAI 2.124, Austin, TX 78712

177. Mary Vernon, Computer Sciences Department, University of Wisconsin, 1210 W. Dayton Street, Madison, WI 53706

178. Robert G. Voigt, National Science Foundation, Room 417, 1800 G Street N.W., Washington, DC 20550

179. Harald Wabnig, University of Vienna, Institute for Statistics and Computer Science, Lenaug. 2/8, A-1080 Vienna, Austria

180. Thomas Wagner, Computer Science Department, Vanderbilt University, Nashville, TN 37235

181. Tammy Welcome, Lawrence Livermore National Lab, Massively Parallel Computing Initiative, L-416, P. O. Box 808, Livermore, CA 94550

182. Steve Weston, Scientific Computing Associates, Inc., One Century Tower, 265 Church St., New Haven, CT 06510

183. Mary F. Wheeler, Department of Mathematical Sciences, Rice University, P. O. Box 1892, Houston, TX 77251

184. Andrew B. White, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545

185. John Zahorjan, Department of Computer Science and Engineering, Sieg Hall, FR-35, University of Washington, Seattle, WA 98195

186. Roland Zink, Institute of Parallel and Distributed Supercomputers, University of Stuttgart, Univ. Stuttgart, IPVR, Breitwiesenstrasse 20, W-7000 Stuttgart 80, Germany

187. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P. O. Box 2001, Oak Ridge, TN 37831-8600

188–197. Office of Scientific & Technical Information, P. O. Box 62, Oak Ridge, TN 37831

END

DATE
FILMED
12/7/92