Extensions of MEDEA for the performance analysis of HPF+ benchmarks

M. Calzarossa, L. Massari, A. Merlo, D. Tessera
Dipartimento di Informatica e Sistemistica
Università di Pavia
Via Ferrata 1, I–27100 Pavia, Italia

Deliverable: D2.3a
Date: March 1997
Status: draft
Confidentiality: public
Responsible: University of Pavia

Revised: April/7/1997

The **HPF+** Consortium:
AVL GesmbH - Austria
European Centre for Medium-Range Weather Forecasts (ECMWF) - England
Engineering System International SA (ESI) - France
NA Software Ltd (NAS) - England
University of Pavia - Italy
University of Vienna - Austria

Contents

1 Introduction

This report describes the enhancements and the extensions introduced into MEDEA with the aim of evaluating the kernel benchmarks developed within the HPF+ project.

The evaluation relies on an experimental approach based on the analysis of the actual performance achieved by the benchmarks. Measurements are collected at run–time by monitoring the executions of the kernels on the parallel system and trace files are produced at the end of their executions. MEDEA processes these trace files and derives metrics and performance parameters able to capture and to explain the behavior of the kernels.

Two complementary monitoring approaches are used to trace the execution of the HPF+ kernels. The Vienna Fortran Performance Measurement System (VFPMS), which is part of the VFCS environment [1], allows the tracing of any portion of the kernels (e.g., loops, subroutines, statements). The profiling libraries provided as part of MPICH [2] intercept all the calls to MPI functions (e.g., point–to–point and collective communications). Note that the calls to MPI functions are hidden into the PARTI/CHAOS libraries [3] which are used by the compiler to manage parallelization and data distribution.
At the end of the execution of the kernels, both VFPMS and MPICH produce their own trace file where detailed information about the intercepted events is stored according to a PICL–like and to the ALOG format, respectively.

As part of the extensions of MEDEA, two filtering modules have been implemented to process VFCS and MPICH trace files. Each of these modules analyzes its corresponding trace file and derives various types of metrics (e.g., signatures, speedup), profiles and performance parameters (e.g., execution time, communication time).

Browsing through metrics, profiles and parameters is possible by means of the `Visualization` and of the `Navigate` modules of MEDEA. In particular, in order to ease the interpretation of the performance and of the behavior of HPF+ kernels, the `Visualization` module has been enhanced by introducing various facilities, such as, zooming.

Another important feature of MEDEA discussed in this report is the "automatic run" mode. This feature was introduced because of the requirements application developers raised during the "Training on MEDEA" [4] organized by the University of Pavia in July 1996. Instead of having the users of MEDEA to provide the specifications of the analysis (e.g., metrics and parameter selection) via MEDEA graphical interface, it is possible to start an automatic run. This means that MEDEA processes the specified set of trace files and produces all possible results (e.g., metrics, performance parameters) which are then available to be browsed and visualized. The automatic run mode can be profitably used to have some sort of preliminary overview of the behavior of the kernels; these results can then drive further refinements of the analysis.

The report is organized as follows. Section 2 presents an overview of MEDEA by pointing out its main characteristics and its role within the project. The tracing facilities used within the project are described in Section 3. The various extensions and enhancements of MEDEA are presented in Sections 4 and 5. Section 6 discusses how MEDEA has been used during the first year of the project and outlines future extensions of the tool. The Appendices describe in more details various technicalities related to tracing issues (e.g., format of trace files, tracing setup) and to the use of MEDEA (e.g., automatic run mode).

2 Overview of MEDEA

This section presents the basic structure and the main characteristics of MEDEA together with its role within the project. Sections 4 and 5 will describe the extensions implemented within the project.

MEDEA [5], [6] is a software tool developed at the University of Pavia for the analysis of parallel programs. The tool has been extended and enhanced within the project for the evaluation of the kernel benchmarks with the aim of providing synthetic descriptions of their behavior which summarize their most relevant characteristics. The tool has been profitably used for the evaluation of the version 1 of kernel benchmarks (see [7], [8]).

Among the possible approaches to the performance evaluation of the parallel programs, a "measurement–based" approach has been adopted within MEDEA. Hence, measurements collected at run–time during the execution of the programs on parallel systems represent the basis of the evaluation.
Depending on the objective of a particular study, measurements can be collected at various levels of details: full program, task, subroutine, loop, or arbitrary code segment. The selected level identifies the program component, driving the corresponding monitoring activity. The code has to be instrumented with the insertion of measurement points associated to the components of interest. The program execution has then to be monitored and events corresponding to the entry and exit points from each instrumented component have to be collected. MEDEA then processes the trace files produced at the end of the execution.

MEDEA combines in a user–friendly environment with a graphical interface based on Motif, the latest techniques for workload characterization [9], [10]. Figure 1 shows the interaction of MEDEA with monitoring systems, and its role within the project.

The tool has been structured in a modular way and each module addresses specific issues related to the analysis of the parallel programs. This structure allows also an easy integration of new functionalities. Depending on the objectives of the study, a few of these modules are used in combination with each other. Each module provides different results which give insights into particular aspects of the behavior of a parallel program and which represent, at the same time, the basis for the analyses applied within other modules.

MEDEA provides various types of statistical and numerical techniques for the analysis of parallel programs. By processing trace files, the tool derives a set of parameters (e.g., computation and communication times) and performance metrics (e.g., profiles and speedup). During the preliminary analysis, statistical techniques are applied. For each parameter, basic statistical indices, such as, mean, variance, coefficient of variation, and percentiles are computed in order to give preliminary insights into the program behavior. In particular, by looking at the computed percentile values, outliers, which correspond to program components characterized by parameter values very different from their "average' can be identified. For example, an outlier can be a processor whose communication time is much smaller (or greater) than the others'.

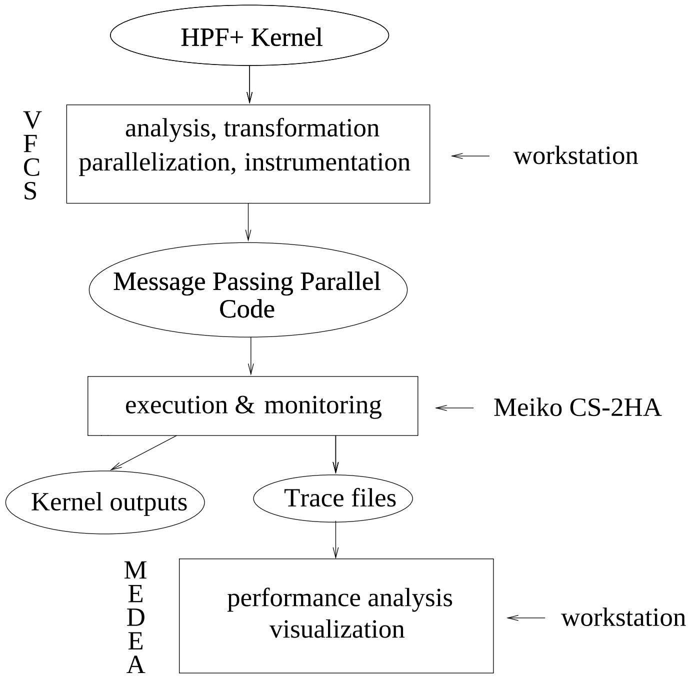


Figure 1: Role of MEDEA in the evaluation of the project benchmarks.

Clustering can then be applied when some sort of classification has to be obtained. Cluster analysis identifies homogeneous groups of components described by the parameters selected according to the objectives of the study. The components, considered as points in a multidimensional space, are grouped according to their similarities. The result of the cluster analysis is a partition into groups, characterized by their centroids, i.e., their geometric centers. For each of these classes, basic statistics and distributions of the parameters used are provided.

Further analyses can be performed by applying numerical fitting techniques, in order to obtain analytical representations of the dynamic behavior of parallel programs. Experimental measures (e.g., communication profiles, speedup, signatures) can be summarized by means of analytical expressions which can be simple functions, like polynomials, trigonometric functions, exponentials, or combinations of a few of them.

3 Tracing

HPF+ kernel benchmarks, parallelized and instrumented by means of the VFCS, are monitored and a set of measures to be analyzed by MEDEA is collected at run–time (see Fig. 1). In particular, depending on the type of instrumentation adopted, two tracing mechanisms, one provided within VFCS and one based upon MPICH facilities, can be used. These monitoring approaches are able to capture different sets of program–generated events. When both tracing mechanisms are employed, they collect complementary information which provides comprehensive insights in the benchmark behavior. The basic set of events stored into the trace files by the monitoring systems can be roughly subdivided into the following groups:

- Program components: these events are associated to the entry and to the exit from specific portions of the kernel benchmark, that is, subroutines, loops, arbitrary code segments, or single statements.
- Interprocessor communications: these events are associated to the communication activities of the kernel benchmark. Both point–to–point (e.g., `MPI_Send`, `MPI_Recv`) and collective communications (e.g., `MPI_Gather`, `MPI_Bcast`) are traced.
- Synchronizations: these events are associated to the synchronization activities of the processors allocated to the kernel benchmark. These synchronization points usually correspond to some sort of global communication mechanism (e.g., `MPI_Barrier`).

Note that the tracing mechanism within VFCS provides the possibility of a high–level instrumentation of specific programs components; the MPICH tracing facilities intercepts the events related to interprocessor communications and synchronization activities.
In what follows, more details about the two tracing mechanisms adopted to monitor the execution of HPF+ kernel benchmarks are presented.

3.1 VFCS tracing

VFPMS is the monitoring system provided as part of the VFCS environment. The instrumentation of the code and the tracing facility are the two main components of VFPMS.
VFCS graphical interface [11] allows the access to VFPMS, for the instrumentation of HPF+ kernel benchmarks. The user has to select the program components of interest (e.g., subroutines, loops, arbitrary code segments). The specification of these components takes place after the parallelization and optimization phases applied to the HPF+ kernel benchmark, and before the parallel code for a specific parallel architecture is generated. VFPMS derives, depending on the specifications, the corresponding measurement points and inserts calls to the `TIMEAS` routine, which is part of the library within VFPMS responsible for the run–time monitoring activity.
More details about the instrumentation facilities of VFCS will be provided in Appendix B.1.

At the end of the run of an instrumented benchmark on the target parallel system, the VFCS produces a trace file which is used for a post–mortem analysis within MEDEA. Both execution and communication times for the selected components are measured. Note that VFPMS is able to trace only regular communication patterns, that is, point–to–point communications mapped by VFCS into `EXSR` (EXchange Send/Receive) operations.

VFPMS stores the generated trace file according to a PICL–like format [12]. New event types have been added to the standard PICL format to deal with VFCS specific program components. An example of a VFCS trace file is presented in Appendix C.1.

3.2 MPICH tracing

The MPICH profiling libraries used within the project are the monitoring system provided as part of MPICH [2], an implementation of the MPI protocol developed by the Argonne National Laboratory.

These libraries are able to trace, by means of wrapping libraries, each MPI function (e.g., `MPI_Send`, `MPI_Recv`, `MPI_Bcast`) issued by the processors allocated to the kernels. It is important to notice that both MPI functions explicitly inserted by the compiler into the generated code and implicit functions, hidden into the PARTI/CHAOS library, are traced by MPICH.

No instrumentation of the kernels is required. A setup phase is needed in order to enable the MPICH tracing mechanism, which then monitors every MPI function. This setup only requires the specification of a few options for linking the MPICH profiling libraries (see Appendix B.2).

MPICH provides the `MPI_Pcontrol` function to enable/disable the tracing. This facility is very useful, for example, when the amount of information stored into the trace files becomes too large or when the tracing has to be focused on specific portions of the kernels. Appropriate calls to the `MPI_Pcontrol` function have to be inserted, by hand, in the parallelized code produced by VFCS. Details on both the setup phase and the use of `MPI_Pcontrol` are provided in Appendix B.2.

MPICH stores the generated trace file according to the standard ALOG format [2]. An example of a trace file is presented in Appendix C.2.

4 Interface to MEDEA

In what follows, the modules, which have been extended for the analysis of the performance of HPF+ kernel benchmarks, are described. The aim of these enhancements is to provide a detailed description of the behavior of HPF+ benchmarks by processing the collected trace files and to help the application developers in the interpretation of the results provided by MEDEA.

The overall management of the analysis environment of MEDEA has been eased via the introduction of the concept of "session". Two new filtering modules have been implemented in order to allow MEDEA to process the trace file formats adopted within VFCS and MPICH. The visualization facilities have been improved to include new features (i.e., the zooming capabilities) which allow a more intuitive interpretation of the results of the performance analysis.

4.1 Sessions

The concept of "session" has been added to MEDEA as a management facility. A session has been defined as the logical organization of a set of analyses performed on specific trace files. A directory tree is used to store the specifications of the analyses and the results of a session. Specific functionalities aimed at session management have been developed, namely:

- `List`: shows the names of the existing sessions.
- `Open`: restores an existing session (which becomes the current session) and makes available all the specifications and the results of the previous analyses.
- `Rename`: changes the name of a session. By selecting this item, the new name has to be specified.
- `Delete`: removes a session by choosing its name through a menu. The tree of files and directories containing the specifications and the results of the analysis are removed.
- `Close`: closes the current session by saving all the corresponding specifications and results. This action is mandatory before deletion or renaming of a session, before restoring an existing session or opening a new one, as well as before exiting MEDEA.
- `New`: creates a session, whose name has to be specified.

4.2 Filtering

Information stored into trace files are processed by the `Filtering` module of MEDEA for deriving profiles and high–level performance parameters and metrics.
Two new filtering modules have been added to MEDEA in order to process the trace files formats adopted by VFCS and MPICH (i.e., PICL–like and ALOG, respectively). The integration of these filters has been eased because of the modular structure of the tool.

Once the names of the trace files to be analyzed have been specified, parameters, such as execution, computation, and communication times of the kernel, are computed. The number and the type of these parameters depend on the information collected into the trace files.
The set of metrics, derived both from VFCS and MPICH trace files, is:

- speedup: ratio between the execution time on a single processors and the execution time on p processors. Note that two possibilities for specifying the execution time on a single processor are given within MEDEA. The time can be derived from the corresponding trace file, if any, or it can be specified directly by the user of MEDEA on the command line (see Appendix D);
- efficiency: ratio between the speedup (computed with p processors) and the number of processors p;
- efficacy: ratio between the square of the speedup (computed with p processors) and the number of processors p;
- signatures: total execution, computation, and communication times spent by the kernel as a function of the number of allocated processors. Note that, from MPICH trace files, only execution and communication signatures are derived. Furthermore, for VFCS trace files, this metric is meaningful if the kernel benchmark performs regular communications only.

Profiles represent the number of processors performing a specific operation (e.g., communication, send, receive) as a function of the execution time of a program. Note that the naming formalism adopted for the profiles derived by MEDEA is `<trace-name>.<type>`, where the `<type>` field identifies the contents of the profile (e.g., the communication protocol).

In the following sections, the results that can be obtained from the VFCS and MPICH filtering modules are described.

4.2.1 VFCS filtering

The information collected in the VFCS trace files strongly depends upon the instrumentation level selected via the VFPMS interface. Trace files can contain information that refers to the whole kernel or to its individual components (i.e., subroutines, loops, arbitrary code segments and single statements).

The set of possible parameters, and their corresponding mnemonic names used within MEDEA, are the execution (`t_exec`), computation (`t_comp`), communication (`t_comm`), send (`t_send`), and receive (`t_recv`) times, the volume of data exchanged among processors (`d_comm`), the volume of data sent (`d_send`) and received (`d_recv`), the number of regular communication (`n_comm`), the number of send (`n_send`) and receive (`n_recv`) operations. When the trace files are analyzed at the "program" level, the values of these parameters represent the average computed with respect to the number of allocated processors. When the trace files are analyzed at different levels of detail (e.g., loop), these values represent the overall timings and frequencies of each monitored component on each processor.
Note that, depending on the measurement points inserted by VFPMS and to the events that trigger at run–time the measurement system, MEDEA automatically produces a few files, each containing the parameters that describe each component. The names associated to these

files contain a fixed suffix, added to the name of the analyzed trace file, that conventionally identifies the level of the analysis. These extensions are summarized in Table 1.

Profiles express the number of processors involved in execution, computation, regular communication, send, and receive operations, as a function of the total execution time of the kernel benchmark. Note that regular communications are implemented by means of `EXSR` operations, which correspond to blocking send–blocking receive pairs. As a consequence, the communication profile for `EXSR` operations can be perfectly split into its send and receive components.

.Prog	Whole kernel	**.Loop**	Loops
.Proc	Processors	**.Code**	Arbitrary code segments
.Sbrt	Subroutines	**.Stat**	Arbitrary statements

Table 1: File suffixes adopted within MEDEA for the various components.

4.2.2 MPICH filtering

The traces collected by means of the MPICH profiling libraries can be analyzed by MEDEA for deriving profiles, metrics and performance parameters, able to explain the behavior of the whole program (i.e., kernel) and of the individual processors. Filtering of MPICH trace files is based on two possible levels of characterization, i.e., "program" and "task". The main difference between these two levels is represented by the performance parameters computed. The parameters produced by the analysis at the "program" level represent the average times spent by the processors for performing each of the MPI functions used within the kernel together with their average number of calls. For example, `t_Bcast` and `n_Bcast` represent the average time spent by each processor in `MPI_Bcast` function and their average number of calls. The "task" level analysis provides per processor parameters, that is, the overall time spent by each processor in each MPI function and its frequency are computed.

MEDEA automatically produces a few files containing the parameters that describe each component. The names associated to these files have a fixed suffix (i.e., **.Prog** and **.Proc**) which conventionally identifies the level of the analysis and is added to the name of the analyzed trace file.

Moreover, for the analysis at the "program" level MEDEA produces metrics and profiles. Each MPI operation is characterized by its profile, that is, the number of processors simultaneously performing the specific operation as a function of the execution time. MEDEA produces additional profiles with the objective of summarizing the overall behavior of the communications events. In particular, we have:

- communication profile: represents, as a function of the execution time, the number of processors simultaneously involved in MPI operations;

- transmission profile: represents, as a function of the execution time, the number of processors which are simultaneously sending messages, i.e., performing any type of MPI operations referring to point–to–point send operations (e.g., `MPI_Send`, `MPI_Bsend`, `MPI_Rsend`, `MPI_Ssend`);
- receiving profile: represents, as a function of the execution time, the number of processors which are simultaneously receiving messages, i.e., performing any type of MPI operations referring to point–to–point receive operations (e.g., `MPI_Recv`, `MPI_Irecv`, `MPI_Recv_init`);
- collective profile: represents, as a function of the execution time, the number of processors simultaneously involved in one–to–many or many–to–many operations (e.g., `MPI_Allgather`, `MPI_Allreduce`, `MPI_Alltoall`, `MPI_Barrier`, `MPI_Bcast`).

4.3 Visualization

The `Visualization` module of MEDEA provides an interface to display, on a session basis, all the results produced by the various modules of the tool.
As can be seen from Figure 2, the results are graphically grouped according to the analysis they come from, namely, filtering, preliminary analysis, clustering and fitting. When some results are not available because they have not been requested or because they are meaningless in a specific context, the corresponding buttons are insensitive and displayed with a ghosted tile.

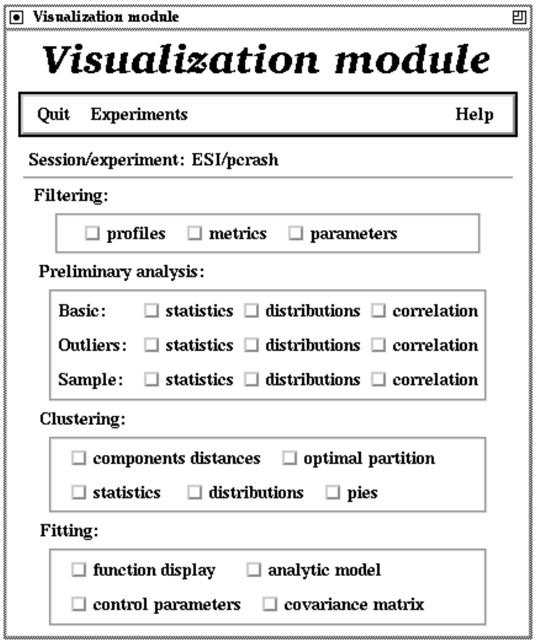


Figure 2: Main window of the `Visualization` module.

The extensions of the `Visualization` module, which mainly deal with the `Filtering`, are presented in what follows:

- profiles and metrics: by clicking the `profiles` or `metrics` buttons, a window is presented, containing a list of buttons for the selection of individual profiles or metrics, respectively. Note that the type of results obtained depends on the contents of the trace files and on the level of analysis performed.
  The signatures of the version 1 of the HPF+ kernel by AVL, executed varying the number of processors from 3 up to 33, are shown in Fig. 3. Figure 4 shows an example of visualization of the communication profile of the same kernel, executed with 17 processors.

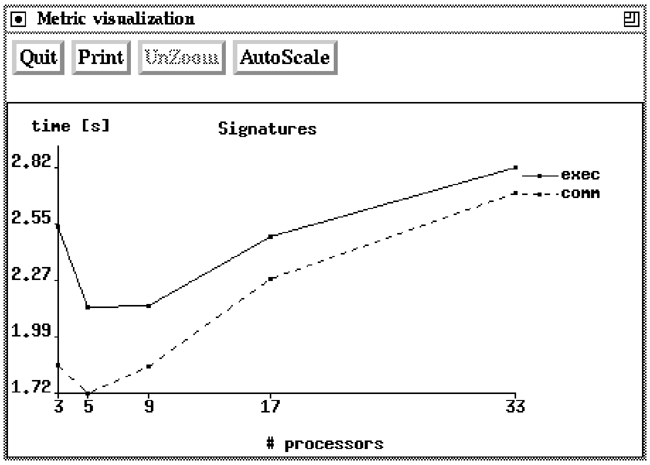


Figure 3: Signatures of version 1 of the HPF+ kernel by AVL.

- parameters: when this item is activated, a window is popped up where the names of the parameter files produced by MEDEA can be selected, displayed or printed.

The visualization module has been extended by providing zooming facilities, to highlight specific portions of the various diagrams produced by MEDEA. Figure 6 shows the result of the zooming over a portion of the profile of Fig. 5, which refers to the `MPI_Allreduce` function performed by the 33 processors allocated to the kernel by AVL.

Multiple levels of zooming are allowed. By clicking the `UnZoom` button, the original curve is redrawn. Note that, after a zooming, the y–range of the diagram is automatically rescaled according to the actual values. The `AutoScale` button allows the rescale of the y–axis according to the whole range of values and not only to those within the zooming area.

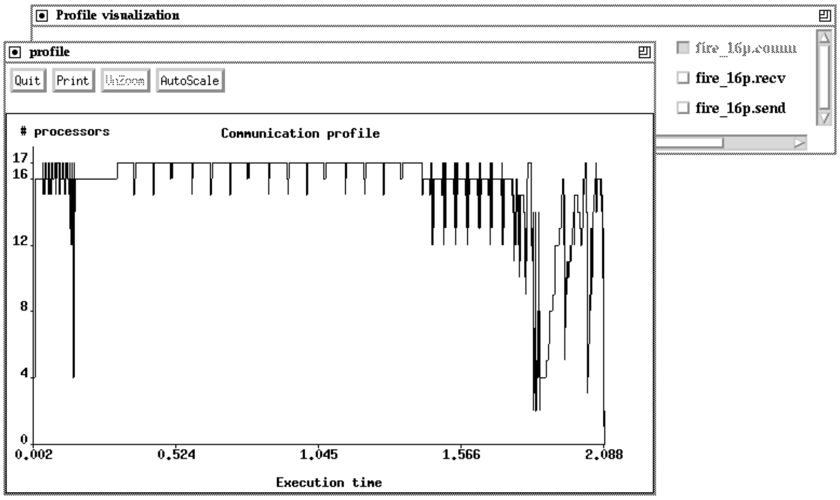


Figure 4: Selection and visualization of the communication profile of the HPF+ kernel by AVL, executed with 17 processors.

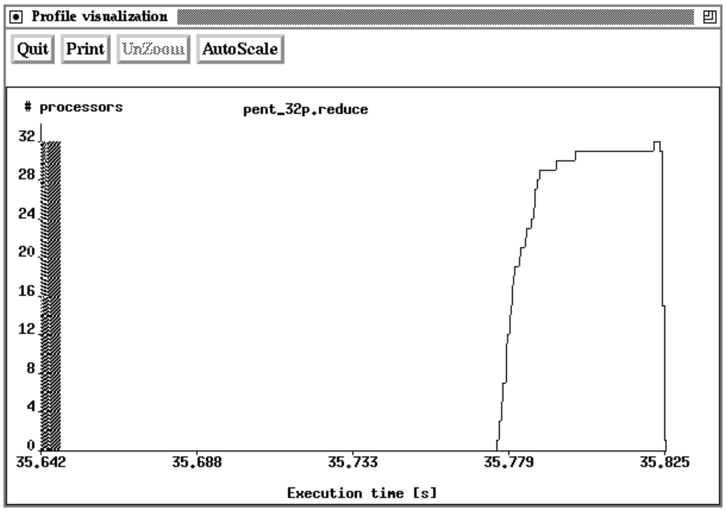


Figure 5: Profile of the `MPI_Allreduce` function and selection of a portion to be zoomed.

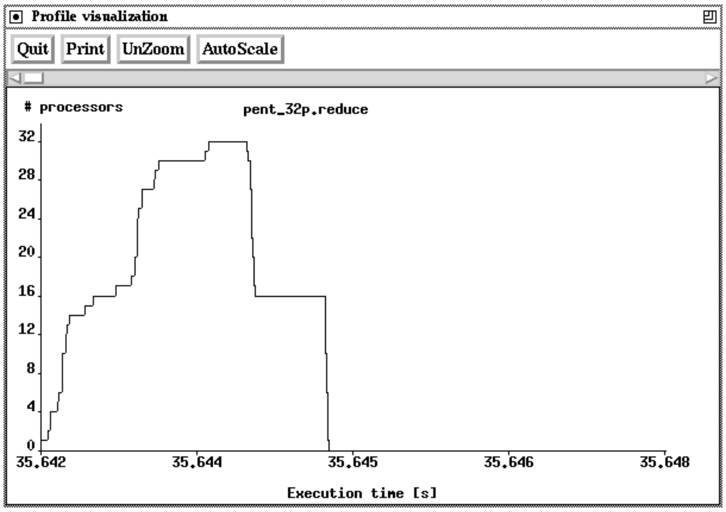


Figure 6: Portion of the profile of Fig. 5, after zooming.

4.4 Navigate

This module allows the browsing through text files, such as trace files, logs of the performed analyses, and parameter files produced by MEDEA.
When this module is invoked by clicking the corresponding button of the main window of MEDEA, a window containing the list of the directories to navigate through and of the files to browse, is presented (see Fig. 7).

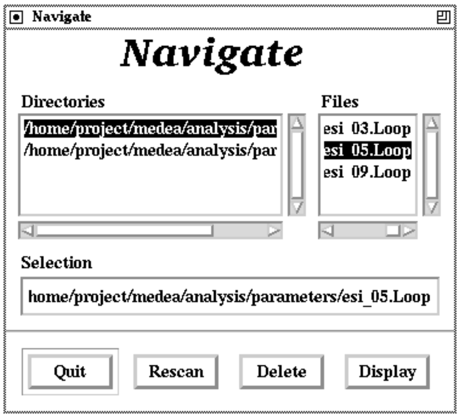


Figure 7: Main window of the "`Navigate`" module.

Besides the `Quit` and `Rescan` operations, once a file has been selected, it is possible to delete it or to display it. If the `Display` button is clicked, a window is popped up and the file can be edited, saved on disk or printed.

5 How to run MEDEA

MEDEA can be used either in an interactive mode or in an automatic mode. Details on how to run the tool are provided in the following sections.

5.1 Interactive run

In the interactive mode, the user of MEDEA has to provide, by using its graphical interface, all the specifications which will drive the analysis. In particular, the trace files to be filtered, the metrics and profiles to be derived are to be selected via the graphical interface. The command

```
xmedea
```

starts the main window of MEDEA. In such a case, the user can access the `Navigate` and `Help` modules only. In order to access the analysis modules, a session has to be created from scratch or an existing one has to be restored.
The session name can be specified directly on the command line. The command

```
xmedea -session AVL
```

starts MEDEA and creates a new session, named "`AVL`", or restores it if it already exists.

5.2 Automatic run

In order to ease the usage of MEDEA, an "automatic run" mode has been introduced. This facility provides a "non interactive" mode of using the tool. The user intervention is minimal and the graphical interface of MEDEA can be displayed at the end of the analysis only.

The automatic run mode requires the specification of the name of the trace files to be analyzed and is driven by default specifications aimed at providing all the possible results which can be obtained by analyzing the trace files. Based on the contents of the trace files, filtering takes place for each of the recognized levels (e.g., program, loop), and performance parameters, metrics, and profiles are computed.

For example, with the command

```
xmedea -session ESI -trace pcrash_32
```

MEDEA automatically processes the VFCS and MPICH trace files produced at the end of an execution with 32 processors. The names of these trace files must contain the "`pcrash_32`" string. Results are saved into the session named "ESI". Note that, in the automatic run mode, sessions cannot be overwritten. If an existing session name is specified, the automatic run will stop with an error.

When multiple trace files produced by different runs have to be analyzed, MEDEA has to be executed with the `-mtrace` option. With the command

```
xmedea -mtrace avl
```

MEDEA automatically processes all the trace files whose name contains the string "`avl`". Since no session name has been specified, a session named "`default`" is created.

Appendix D explains in more details how this mode works and all the possible command line options recognized by MEDEA.

Note that, in order to avoid duplications, the VFCS filter computes the execution, computation and regular communication profiles only while the MPICH filter produces all the metrics and the remaining profiles.

6 Discussion

The extensions and enhancements introduced within MEDEA revealed particularly useful. MEDEA has been profitably used for the evaluation of the performance of the version 1 of the kernel benchmarks (see [7], [8]). Both the HPF+ kernels and their explicit message–passing versions have been analyzed. The various types of behaviors identified by these analyses suggested the application developers possible improvements of their codes.

Note that the analysis of pure MPI kernels and the comparisons of their performance with the performance of their HPF+ counterparts were made possible because of the development within the project of a filtering module for MPICH trace files. Indeed, as already pointed out, it was not possible to trace by means of VFPMS the irregular communications used within HPF+ kernels since these communications are hidden into the PARTI/CHAOS libraries used by the compiler for parallelization and data distribution. Hence, it was necessary to develop and to integrate into MEDEA a specific module for the analysis of MPICH trace files. In such a way, a comprehensive monitoring activity (i.e., based on both VFPMS and MPICH) of the kernel executions can be performed.

Moreover, during the first year of the project, MEDEA has been successfully ported and made available to various platforms, i.e., SUN, IBM, HP and SGI workstations.

The "automatic run" mode of MEDEA is another enhancement provided within the project whose objective is to provide the full set of all the possible results obtained by processing the trace files. These results give preliminary insights into the behavior and the performance of the kernels and can be used as a basis for further refinements of the analyses.

Future work will be dedicated to provide a functional description of the performance results produced by MEDEA. In such a way, better understanding of the behavior of the kernels will be obtained.

References

[1] S. Benkner, K. Sanjari, V. Sipkova, and B. Wender. Extension of VFCS for the Compilation of Benchmark Programs - Version 1. Document HPF+, D2.2a, Institute for Software Technology and Parallel Systems, University of Vienna, Dec. 1996.

[2] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Technical Report MCS-P567-0296, Argonne National Laboratory, July 1996.

[3] R. Das and J. Saltz. A manual for PARTI runtime primitives. Technical Report, University of Maryland, 1995.

[4] University of Pavia. Training on MEDEA. HPF+ Working Paper, July 1996.

[5] M. Calzarossa, L. Massari, A. Merlo, and D. Tessera. *MEasurements Description, Evaluation and Analysis Tool – User's Guide*, 1997. ftp://mafalda.unipv.it/pub/Docs/Medea-Project/UserGuide.ps.gz.

[6] M. Calzarossa, L. Massari, A. Merlo, and D. Tessera. Parallel Performance Evaluation: the MEDEA Tool. In H. Liddel, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *Proc. HPCN Europe '96*, LNCS 1067, pages 522–529. Springer–Verlag, 1996.

[7] Report on Evaluation Results with Benchmark Programs Version 1. Document HPF+ D1.2a, Dec. 1996.

[8] A. Merlo. Performance Evaluation of HPCN Applications. In *Proc. HPCN Europe '97*, 1997. to appear.

[9] M. Calzarossa and G. Serazzi. Workload Characterization: A Survey. *Proc. of the IEEE*, 81(8):1136–1150, 1993.

[10] M. Calzarossa and L. Massari. Measurement-Based Approach to Workload Characterization. In G. Haring, R. Marie, and G. Kotsis, editors, *Performance and Reliability Evaluation*, pages 123–148. Oldenbourg Verlag, 1994.

[11] S. Benkner, P. Brezany, B. Chapman, V. Sipkova, H. Zima, and al. Vienna Fortran Compilation System – User's Guide. Technical Report, Institute for Software Technology and Parallel Systems, University of Vienna, 1996.

[12] P.H. Worley. A New PICL Trace File Format. Technical Report TM-12125, Oak Ridge National Laboratory, 1992.

A Trace naming scheme

In order to ease the analysis of HPF+ kernels, based upon the VFPMS and MPICH monitoring systems, a trace naming formalism has been adopted. By default, the VFPMS and MPICH tracing mechanisms have a standard naming scheme for the collected trace files. A trace file produced by VFPMS is named tracefile.trf, while a trace file produced by MPICH is named <executable>_ profile.log. As a consequence, after each execution on the Meiko CS–2, to avoid confusion, it is up to the user to rename the generated traces according to the formalism adopted within the project. In the case of VFCS and MPICH trace files, the naming scheme recognized by MEDEA is

<program>_<processors>.<type>

A trace file name consists of three components separated by the "_" sign and by the "." sign, respectively. Note that the "_" and "." signs are reserved and cannot be used in the <program> or <processors> strings. The <program> field contains whichever string may be useful to identify the trace file. The <processors> field contains the number of processors allocated to the program run. The <type> field can be either mpi or vfcs to identify a MPICH or a VFCS trace file, respectively.

As an example, consider the execution of the PAM–CRASH benchmark kernel, with 16 allocated processors. If both VFCS and MPICH tracing have been used, at the end of the kernel execution, the tracefile.trf and node_profile.log trace files have to be renamed according to the previously described formalism. The program name could be "pcrash" and the resulting trace file names could be pcrash_16.vfcs and pcrash_16.mpi, respectively.

B Tracing setup

Both VFCS and MPICH require a setup phase in order to be able to collect at run–time measurements on the behavior of the kernels. The following sections describes how to use the interactive environment provided by VFCS (i.e., VFPMS) and how to enable/disable the MPICH profiling libraries.

B.1 VFCS setup

As already pointed out, VFPMS allows the specification of the set of measurements to be collected during the execution of the kernels. The VFPMS is accessed via the VFCS graphical interface and has to be used in combination with the *Select unit* to select the portions of code to be instrumented. Figure 8 shows the unit browser together with the VFCS main window.

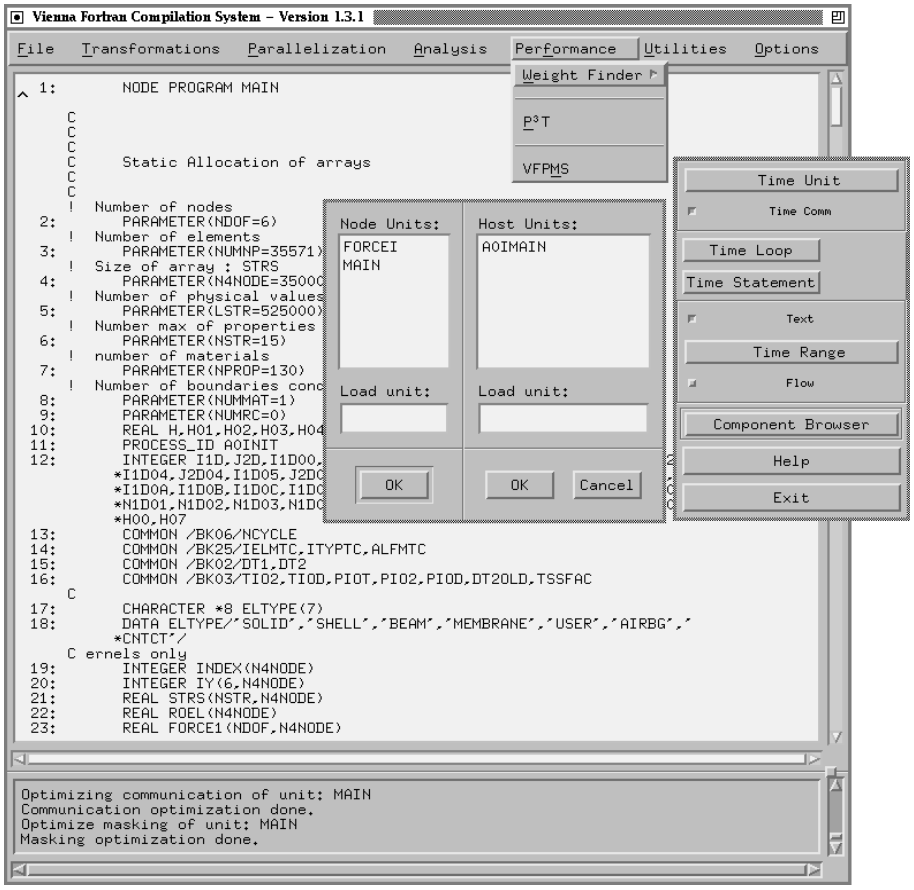


Figure 8: *Select unit* browser and main window of VFCS.

The unit browser loads into the display area of the VFCS graphical interface the main components of the kernels, as identified by means of call graph analysis techniques. In such a way,

it is possible to instrument the whole program (that is, `MAIN`) and each separate high–level subroutine. Once the desired unit is displayed, it is possible to specify the information to be measured at run–time, that is, its execution time and the time spent in regular communications. The `Time Comm` button in the VFPMS window has to be activated, whenever regular communications have to be monitored. The measurement points and the corresponding calls to the `TIMEAS` function of the VFPMS library are inserted at the entry and at the exit point of the actual unit by clicking the `Time unit` button.
Loops, arbitrary code segments, and single statements can be instrumented by means of the `Time Loop`, `Time Range`, and `Time Statements` buttons of the VFPMS window, respectively. In such a case, a message box is displayed and the selection, for example, of the `DO` statement that marks the entry of the loop to be monitored has to take place. Once more, these actions result in calls to the `TIMEAS` function.

B.2 MPICH setup

The MPICH tracing facilities are completely transparent to the VFCS environment. Hence, the tracing is enabled by explicitly linking, by hand, the profiling libraries to the kernel before the compilation phase. This requires a modification of the `Makefile` produced by VFCS. In particular, the "`-lfmpi -llmpi -lpmpi -lm`" options have to be added, just before the "`-lmpi`" option, into those lines that define the `HOSTF77_MEIKO_MPI` and `HOSTLIB_MEIKO_MPI` variables (see Fig. 9).

MPICH linking options

```
#
# MACHINE - Meiko CS-2
# COMMUNICATION LIBRARY - MPI
HOST_MEIKO_MPI = /opt/local/mpich
HOSTF77_MEIKO_MPI = f77
HOSTLD_MEIKO_MPI = f77
HOSTFFLAGS_MEIKO_MPI = -O3
HOSTLIBPATH_MEIKO_MPI = -L$(HOME_MEIKO_MPI)/lib/meiko/meiko -L/opt/MEIKOcs2/lib
HOSTLIB_MEIKO_MPI = -lfmpi -llmpi -lpmpi -lm -lmpi -lew -lelan

NODEF77_MEIKO_MPI = f77
NODELD_MEIKO_MPI - f77
NODEFFLAGS_MEIKO_MPI = -O3
NODELIBPATH_MEIKO_MPI = -L$(HOME_MEIKO_MPI)/lib/meiko/meiko -L/opt/MEIKOcs2/lib
NODELIB_MEIKO_MPI = -lparti -lfmpi -llmpi -lpmpi -lm -lmpi -lew -elan
```

Figure 9: Modification of the `Makefile`, produced by VFCS, for enabling tracing.

The `-lfmpi` option links the kernel code with the Fortran stubs for the MPI functions. The `-llmpi` and `-lpmpi` options provide the MPICH logging and profiling wrappers. Finally, the `-lm` option links the kernel code with the mathematical library. The compilation will then produce an instrumented version of the parallel program which automatically monitors, at run-time, all the MPI functions, including point-to-point and collective communications.

It is important to notice that tracing the whole program may be unfeasible for very complex and iterative codes which issue millions of communication statements. In such a case, it might be impossible to deal with the amount of information to be stored into the trace files. When

these allocation problems arise, or when the performance analyses have to focus on particular aspects of the kernel behavior, it is desirable to restrict the monitoring activity to specific portions of the code.
The MPICH library provides the `MPI_Pcontrol` function, which enables/disables tracing. The calls to this function have to be inserted, by hand, into the parallelized code produced by VFCS, namely into the `node.f` and `host.f` files.
This function initializes the tracing system by enabling the monitoring activity whenever required. The statement

```
call MPI_PCONTROL(1,ierr)
```

enables the monitoring, while the statement

```
call MPI_PCONTROL(0,ierr)
```

switches it off. Figure 10 shows an example where both VFCS (i.e., `call TIMEAS`) and MPICH (i.e., `call MPI_PCONTROL`) monitoring facilities are used.

enables MPICH tracing

```
      call MPI_PCONTROL(1,IERR)
      IF (ASB(1,12) .EQ. 1) CALL MPI_RECV(L,20784 + 16,MPI_BYTE,
     *MPI_ANY_SOURCE,15 + ARS,MPI_COMM_WORLD,status,err)
      call MPI_PCONTROL(0,IERR)
      L = ASB(1,1)
      R = ASB(1,3)
      J01 = 1
      call TIMEAS(1,0,(-1),(-1))
      DO 104 AI01=L,R
         BS(AI01 - AOB(2,1) + 1) = HELP00(J01)
         J01 = J01 + 1
104   CONTINUE
      call TIMEAS(2,0,(-1),(-1))
```

disables MPICH tracing

Figure 10: Enabling/disabling the MPICH monitoring.

In order to have the tracing mechanism disabled at the very beginning of the kernel, the statement

```
call MPI_PCONTROL(0,ierr)
```

has to be inserted in the `node.f` file right after the `call vf_startup()`, as shown in Fig. 11. Once these control directives have been inserted, the parallelized VFCS code can be compiled and executed on the target architecture.

disables MPICH tracing

```
 program node
 INCLUDE 'vf.inc'
 COMMON /A0SPLIT/A0RS,A0SS,A0INIT
 integer A0INIT
 call vf_startup(16)
 call MPI_PCONTROL(0,ierr)
 A0INIT = vf_myhost()
 if(vf_mynode().eq.A0INIT)
*then
      call A0IMAIN
 else
      call MAIN
 endif
 call vf_finalize
 stop
 end
```

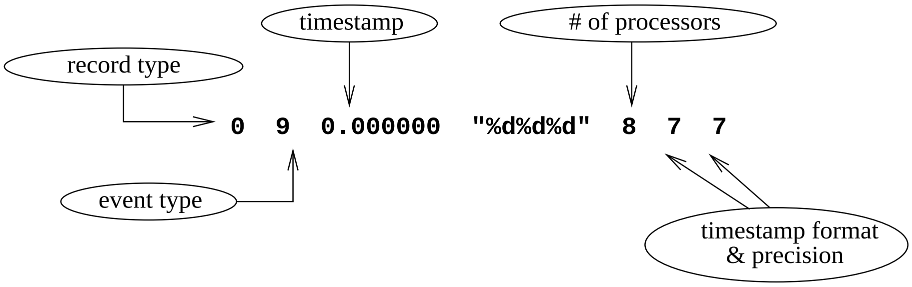
Figure 11: Setup procedure to disable the tracing at the beginning of the kernel run.

C Trace file formats

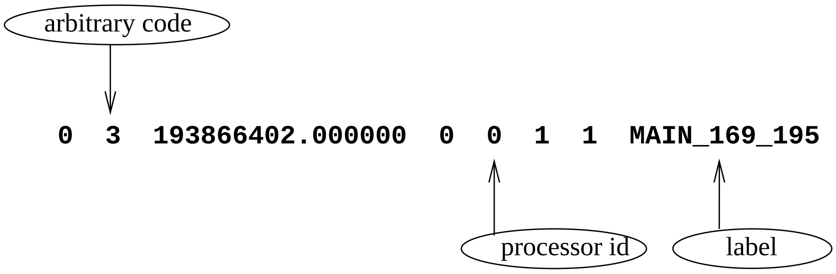
In the following sections, the contents and the format of VFCS (i.e., PICL–like) and MPICH (i.e., ALOG) trace files are described.

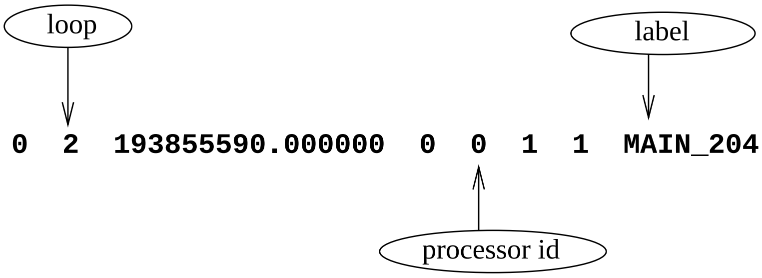
C.1 PICL–like format

The information collected into the trace files strongly depends on the instrumentation level selected while generating the code from VFCS. Each record contains at least six fields. The first three are used to uniquely identify the record type, the event type, and the time stamp. The first record of the trace files is a declaration record (record type `0`) which contains an information event (event type `9`) that summarizes the number of allocated processors and the time stamp format and precision. In our example, the times are expressed in tenths of μs.

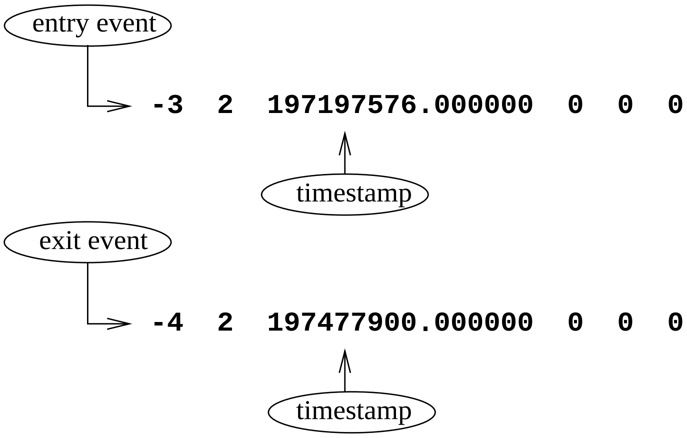


Other declaration records are used to specify the components of the kernels monitored at run–time. As an example, the following records refer to an arbitrary code segment and to a loop.





Each declaration record is followed by a pair of entry/exit records (record type `-3` and `-4`, respectively) that are used by MEDEA to compute the amount of time spent on the specific processor while executing that particular component of the kernel. In our example, the time spent by processor `0` in the loop `MAIN_204` is equal to 280324 μs.



An example of VFCS trace file related to an execution of the HPF+ version 1 of the kernel developed by ESI is shown in Fig. 12.

Note that, record types `-101`, `-102`, and `-103` represent the so called "statistics" records. They summarize the performance figures associated with each component that has been instrumented and monitored.

```
0 9  0.000000 0 "%d%d%d" 8 7 7
-3 -901  0.000000 0 0 0
0 3 192730940.000000 0 0 1 1 MAIN_169_195
-3 3  192730940.000000 0 0 0
-4 3  192732170.000000 0 0 0
0 2 192732890.000000 0 0 1 1 MAIN_204
-3 2  192732890.000000 0 0 0
-4 2  193109902.000000 0 0 0
0 3 193866402.000000 0 0 1 1 MAIN_169_195
-3 3  193866402.000000 0 0 0
-4 3  193866472.000000 0 0 0

..........

0 2 196755590.000000 0 0 1 1 MAIN_204
-3 2  196755590.000000 0 0 0
-4 2  197047406.000000 0 0 0
0 3 197197102.000000 0 0 1 1 MAIN_169_195
-3 3  197197102.000000 0 0 0
-4 3  197197154.000000 0 0 0
0 2 197197576.000000 0 0 1 1 MAIN_204
-3 2  197197576.000000 0 0 0
-4 2  197477900.000000 0 0 0
-4 -901  197635968.000000 0 0 0
-101 -1 197635968.000000 0 0 1 "%d%lf" -901 197635968.000000
-102 -1 197635968.000000 0 0 1 "%d%d" -901 1
-103 -1 197635968.000000 0 0 1 "%d%d" -901 0
-101 -1 197635968.000000 0 0 2 "%d%lf" 2 2804164.000000 3 1694.000000
-102 -1 197635968.000000 0 0 2 "%d%d" 2 9 3 9
-101 2 197635968.000000 0 0 MAIN_204 0
-102 2 197635968.000000 0 0 MAIN_204 0
-103 2 197635968.000000 0 0 MAIN_204 0
-101 3 197635968.000000 0 0 MAIN_169_195 0
-102 3 197635968.000000 0 0 MAIN_169_195 0

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

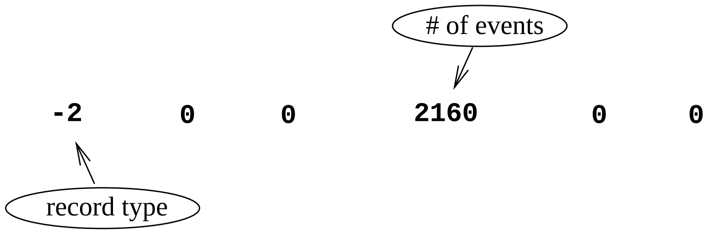
Figure 12: Example of a PICL–like trace file related to an execution of the HPF+ version 1 of PAM–CRASH benchmark developed by ESI.

C.2 ALOG format

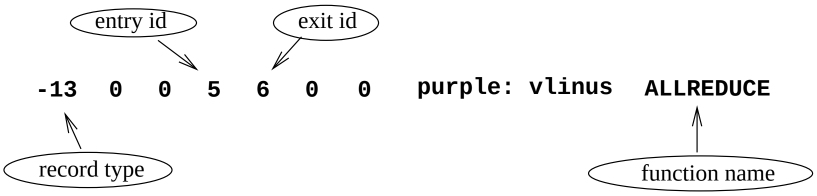
The MPICH trace files are time stamped event trace files. The MPICH profiling libraries intercept each MPI functions (e.g., `MPI_Send`, `MPI_Alltoall`, `MPI_Get_count`) and log into the trace file the events for entry and exit from each function.

The most relevant information contained in each record are the identifiers of the event and of the processor, where the event took place, and the time stamp, expressed in μs, of the event itself.

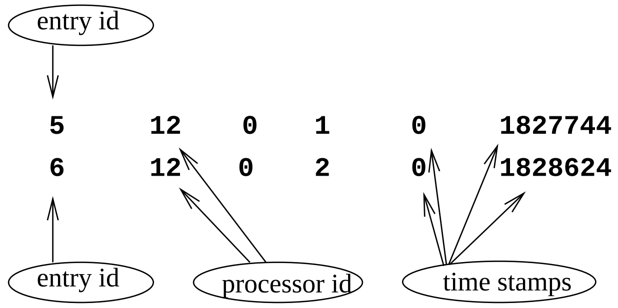
There are a few records which contain a sort of summary information, e.g., number of allocated processors, types of MPI functions. In particular, the number of events stored into the trace file are associated to record type `-2`.



Record types `-13` identify all the MPI functions contained in the trace file and specify the corresponding entry and exit identifiers.



In our case, the `MPI_Allreduce` function is characterized by the events `5` and `6`. An example of entry and exit records, containing the time stamps associated to a call to this function, is shown in what follows. The time spent by processor 12 in this `MPI_Allreduce` is equal to 880 μs.



The number of processors used in the run are stored in the record associated to the event type `-3`. The records `-6` and `-7` identify the time stamps of the beginning and the end of the execution of the program. Figure 13 presents an example of a trace file for an execution of the HPF+ version 1 of the AVL benchmark. The trace refers to an execution with 17 processors. Ten different MPI functions are used during the execution. The global number of calls to MPI functions is equal to 2160.

```
-1 0 0 0 0 0 Me
-2 0 0 2160 0 0
-3 0 0 17 0 0
-4 0 0 1 0 0
-5 0 0 0 0 0
-6 0 0 0 0 11
-7 0 0 0 0 2087588
-8 0 0 1 0 0
-11 0 0 0 0 0
-13 0 5 6 0 0 purple:vlines3 ALLREDUCE
-13 0 9 10 0 0 : ALLTOALLV
-13 0 11 12 0 0 yellow:dimple3 BARRIER
-13 0 13 14 0 0 cyan:boxes BCAST
-13 0 135 136 0 0 : GET_COUNT
-13 0 145 146 0 0 skyblue:gray ISEND
-13 0 153 154 0 0 seagreen:gray PROBE
-13 0 155 156 0 0 green:light_gray RECV
-13 0 161 162 0 0 blue:gray3 SEND
-13 0 213 214 0 0 OrangeRed:gray WAITALL
155 2 0 1 0 2166
155 10 0 1 0 2172
155 14 0 1 0 2204
...............
...............
161 16 0 3 0 45697
-101 16 0 1 0 45712 1 1032
162 16 0 4 0 45862
-102 1 0 16 0 46457 1 1032
156 1 0 2 0 46472
11 1 0 1 0 46493
161 16 0 5 0 51984
-101 16 0 2 0 52001 1 1032
162 16 0 6 0 52155
...............
...............
5 12 0 1 0 1827744
6 13 0 2 0 1828189
6 9 0 2 0 1828213
6 5 0 2 0 1828274
6 1 0 2 0 1828358
6 12 0 2 0 1828624
...............
...............
```

Figure 13: Example of ALOG trace file for an execution of the HPF+ version 1 of the AVL benchmark.

D Command line options

The following command line options are recognized by MEDEA.

`-session <session_name>`	This option is used to specify a session name. If the session does not already exist, it is created. Otherwise, the session with the specified name is restored and all the previous results are made available for visualization or further refinements by means of new analyses. This option can be used in combination with all the following ones.
`-trace <program>_<processors>`	This option is used to identify specific trace files to be automatically analyzed by MEDEA. Whenever invoked, this option forces MEDEA to scan its environment and to look for an MPICH trace file and a VFCS trace file that match not only the `<program>` part but also the `<processors>` one. If the `-n` option is not specified, MEDEA requires a confirmation of the list of identified trace files before starting the automatic run.
`-mtrace <program>`	This option is used to identify the set of multiple trace files to be automatically analyzed by MEDEA. Whenever invoked, this option forces MEDEA to scan its environment and to look for all the trace files that match the specified `<program>` name. If the `-n` option is not specified, MEDEA requires a confirmation of the list of identified trace files before starting the automatic run.
`-n`	With this option, MEDEA disables the confirmations associated with the `-mtrace` and `-trace` options and provides only appropriate error/warning messages upon failures/mismatches detected after the automatic run has started.

`-seq <sequential_time>`	This option is used to specify the execution time of a run of the kernel with one processor. This value, which overwrites the times that could be derived from traces referring to single processor runs, is used for computing speedup, efficiency, and efficacy.
`-quit`	This option prevents the graphical interface of MEDEA to be displayed upon completion of an automatic run. If this option is not specified, at the end of the analysis performed in the automatic run mode the interface is popped up with the current session already active for visualizing the computed results.
`-help`	This option shows the different command line options recognized by MEDEA.