# ANALYZING PICL TRACE DATA WITH MEDEA[†]

*Alessandro P. Merlo*

University of Pavia
Via Abbiategrasso, 209
I–27100 Pavia, Italy

*Patrick H. Worley*

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

**Abstract**

Execution traces and performance statistics can be collected for parallel applications on a variety of multiprocessor platforms by using the Portable Instrumented Communication Library (PICL). The static and dynamic performance characteristics of performance data can be analyzed easily and effectively with the facilities provided within the MEasurements Description Evaluation and Analysis tool (MEDEA). This report describes the integration of the PICL trace file format into MEDEA. A case study is then outlined that uses PICL and MEDEA to characterize the performance of a parallel benchmark code executed on different hardware platforms and using different parallel algorithms and communication protocols.

## 1 Introduction

The demands for hardware and software resources of a computer system significantly influence its performance. Therefore, the quantitative description of resource consumption when running an application plays a fundamental role in every performance evaluation study [CaSe93]. The best way to obtain such a quantitative description for a system is to take measurements while the system is processing its real workload. However, the set of data collected by the monitoring tools represents a detailed "discrete" description of the behavior of the measured applications. While such a characterization is very useful when used as input to visualization tools, it is inappropriate when applied to system modeling, where a compact and manageable representation of the workload processed by the real system is needed.

The process of deriving a compact representation of the workload, *workload characterization*, can be subdivided into several phases [Jain91]. The input to the process is the data collected by monitoring the execution of a given application over the system. Output includes both standard data analysis results, which provide useful insights into the behavior of the application, and workload models, which can be used as input to either simulation or analytic system models. How the data is analyzed and how the model is derived are functions of the type of questions being addressed about the performance of the computer system, the type of data collected, and the level of detail at which the analysis will be performed. For example, at some point in the process, the basic unit of work that is considered in a quantitative description of the workload, the *workload component*, must be specified.

While the type of analysis that is appropriate for a particular workload characterization will vary as different questions are asked or different computer systems evaluated, many mathematical techniques are common to multiple analyses. To support this commonality, researchers at the University of Pavia have developed the MEasurements Description Evaluation and Analysis tool (MEDEA) [MeRo92]. The basic aim of MEDEA is to define an integrated environment in which to perform workload modeling studies. The different operations required to fully examine the behavior of the applications submitted to a system have been logically subdivided into *modules*, each performing a specific manipulation over the performance data and the intermediate results produced at each step of the workload characterization process.

[†]This research was supported by the Applied Mathematical Sciences Research Program, Office of Energy Research, US Department of Energy, under contract DE–AC05–84OR21400 with Martin Marietta Energy System Inc., by the Italian Research Council (C.N.R.) under Grant 92.01571.PF69, and by the Italian MURST under the 40% and 60% Projects.

The collection of performance data is often a difficult task in itself, especially on systems without dependable operating system or hardware support for the collection of useful data. One portable option for the collection of performance data for message passing computer systems is to use the Portable Instrumented Communication Library (PICL), developed at Oak Ridge National Laboratory, when implementing application codes [GHPW90]. PICL implements a generic message–passing interface able to support interprocessor communications on a variety of different hardware platforms. Furthermore, PICL tracing routines allow the user to collect detailed information on the behavior and performance of parallel programs. The trace files generated by PICL can be used as input both to performance visualization tools, e.g. ParaGraph [HeEt91b], for performance tuning and debugging, or to performance evaluation tools.

This report describes the integration of PICL trace data into MEDEA and how the static and dynamic characteristics of the workload generated by PICL applications can be analyzed with the facilities provided within MEDEA. Sections 2 and 3 give a brief description of the main features provided within PICL and MEDEA, respectively. Section 4 deals with the integration of PICL and MEDEA: the selection of possible workload components and the specification of the corresponding performance parameters are outlined here. Section 5 outlines an experimental application. A few conclusions are summarized in §6.

## 2 The Portable Instrumented Communication Library

A detailed performance analysis of the behavior of a computer system under its real workload can be achieved by means of event–driven monitors, i.e., tools that capture the events generated by a program and store them into trace files. However, the trace file formats adopted by different monitoring tools are, in general, quite different from one to another (see, for example, [GHPW90], [HeLu91], [LeSe92]), with each developer defining a specific record format able to address those events of interest for the particular system being evaluated. This lack of standardization makes it difficult to easily analyze trace files collected on different systems, but is a reflection of system differences that can not simply be eliminated by a standardization process. Recently, there has been a movement toward establishing a standard metaformat in which to specify trace file formats [Aydt92]. If this approach is adopted, it will ease one aspect of integrating new types of trace data into tools like MEDEA. But it will not eliminate true semantic differences between the information collected on different systems or with different tools, and the integration of new types of trace information will always require careful thought and design.

The PICL trace file format was chosen for integration with MEDEA because of the wide availability and utility of PICL trace files. The machine independent layer of PICL has proven to be a sufficient framework to support portability between different platforms, and the trace file format used by PICL is flexible enough to collect data for performance evaluation. Moreover, while many of the available traces are generated from PICL programs, a significant number are generated directly by other event-monitoring systems or via postprocessing, so that they can be visualized with ParaGraph.

The PICL trace file format was recently significantly modified, to better support the collection of information found to be most useful in visualization tools like ParaGraph and in workload characterization, and to be more extensible [?]. The new trace file format has been incorporated into ParaGraph, and is being considered for use in other event monitoring systems. Because of the significant new capabilities of the new format, and because of its adoption outside of PICL, it is the new format that has been integrated with MEDEA.

The basic structure of PICL trace records is shown in Table 1.

| Record type [int] | Event type [int] | Timestamp [double] | Processor ID [int] | Task ID [int] | Number of data fields [int] | Data descriptor [int or string] | Data |
|---|---|---|---|---|---|---|---|

Table 1: Basic structure of PICL trace records.

Four different record types are currently supported by PICL:

- *user–defined* record types are used to specify the data associated with user–defined events;
- *event* record types are used to collect detailed information, for both system and user–defined events, needed for a visualization tool like ParaGraph or for the analysis of user events by means of MEDEA, as will be explained in §4.1;
- *statistics* record types are used to collect profile data of system and user–defined events;
- *subset–definition* record types are used to define subsets, e.g., of processors or processes, for which cumulative statistics are to be collected.

The process of workload characterization using PICL trace files is based primarily on the analysis of event and statistics record types. The tracing facilities provided within PICL allow the user to specify the amount and the type of data to store into trace files: if detailed data are needed, then for each event generated by the application (e.g., send/receive commands, I/O operations, etc.) timestamped entry/exit records are stored for the processor and the process associated with the event; if only global information are needed (e.g., when it is not important to know the exact timing of the single events but when we are interested in the corresponding cumulative times), then the statistics records can be used to characterize the general behavior of an application.

The event types currently supported by PICL cover most of the event data utilized in performance evaluation studies of message-passing parallel applications. The most important categories of events follow:

- *user–defined* events allow the user to specify that the execution of a subroutine or even arbitrary code segments be considered an event of a certain type, allowing the logical structure of the application to be represented during subsequent analysis;
- *interprocessor communication* events represent PICL commands for enabling, disabling, or invoking interprocess communications, including, for example, send and receive;
- *file I/O* events are used to collect performance data on (physical) I/O, which strongly influence the performance of most real parallel applications;
- *synchronization* events currently supported include "clock normalization" and "barrier";
- *resource allocation* events deal with the allocation/deallocation of processors;
- *tracing* events are recorded with the dual goals of allowing a correct interpretation of the trace files and of providing a measure of the overhead implied by the tracing activity.

## 3 The MEasurements Description Evaluation and Analysis tool

The construction of accurate workload models requires the application of different types of statistical and numerical techniques interacting together to fully characterize the behavior of the applications submitted to a system. During the design phase of MEDEA[1], the need for integration between these different underlying techniques and the need for portability across a variety of computer platforms led to the choice of a standard development environment. As a consequence, MEDEA is currently implemented on UNIX systems running X Windows/Motif [2]. Figure 1 shows the main window of the graphical interface provided by MEDEA.

**Data manipulation module.** The *data manipulation* module performs a preliminary analysis of the trace data in order to correlate the events recorded during the execution of an application. Traditional

[1]For a detailed description of the tool, see [Merl93].
[2]MEDEA requires at least X11R5 and Motif 1.1.4 to work properly.

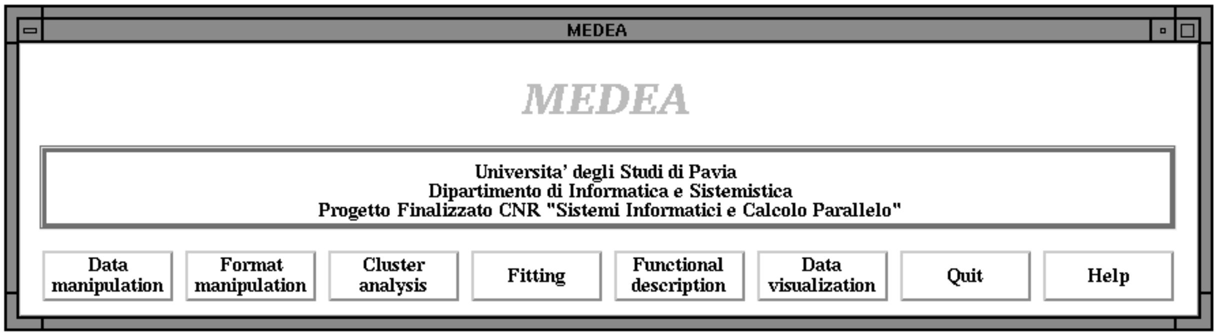
MEDEA
MEDEA
Universita' degli Studi di Pavia
Dipartimento di Informatica e Sistemistica
Progetto Finalizzato CNR "Sistemi Informatici e Calcolo Parallelo"
Data manipulation
Format manipulation
Cluster analysis
Fitting
Functional description
Data visualization
Quit
Help

Figure 1: Main window of the graphical interface of MEDEA.

performance indices, such as computation and communication times, and parallel metrics, such as speedup and efficiency, can then be derived by filtering the trace data. Until recently, MEDEA required that all trace data be in the format generated by the PARallel MONitor (PARMON), a distributed event–driven monitoring tool [LeSe92] for transputer–based MIMD architectures. New trace file analysis facilities have been added to the data manipulation module of MEDEA to parse the PICL trace file format as well.

**Format manipulation module.** Within MEDEA, a *format* is a subset of the performance parameters that can be associated with the specific workload component under study. The *format manipulation* module of MEDEA enables user–defined subsets of parameters to be stored in an internal library, allowing subsequent workload analyses to be performed with fewer interactions with the graphical interface.

**Cluster analysis module.** The *cluster analysis* module is used to examine the statistical properties of the measured data set. Multidimensional cluster analysis techniques are used to identify groups of workload components having homogeneous characteristics with respect to some predefined parameters. The clustering algorithm implemented within MEDEA is the *k–means*, an iterative nonhierarchical method of partitioning data sets [Hart75]. Each partition is derived by minimizing the distances between each workload component and the centroid of the cluster it belongs to. At the end of the analysis, the optimal partitions (if any) are derived according to the overall mean square ratios of the evaluated clusters.

**Fitting module.** Workload models must be compact and easily manageable. The *fitting* module provided within MEDEA allows the user to derive analytic descriptions of the dynamic behavior of the workload from the measured data. The analytic models are described in terms of one or more of the collected parameters, and are able to represent the variations of the workload parameters with respect to any independent variables, including time.

**Functional description module.** The process of workload characterization can be approached from two different viewpoints. The “physical” viewpoint describes the behavior of the system and the applications by means of indices related to resource consumptions, such as computation and communication times. This approach is the one realized by the data manipulation and the cluster analysis modules of MEDEA. The “functional” viewpoint gives a logical description of the workload. In this case, the classification of workload components is based, for example, on the membership of particular components in a specific cluster. The *functional description* module of MEDEA deals with this second viewpoint.

**Data visualization module.** The graphical visualization of the parameter values derived from the trace data and of the results produced by the various analyses performed within MEDEA is often an important tool in understanding the characteristics and the behavior of the workload. The *data visualization* module of MEDEA provides this facility.

## 4 PICL–MEDEA Integration

The selection of appropriate workload components and of the corresponding performance parameters is strongly dependent on the type of information collected into the trace files. Since PICL tracing routines allow the user to specify the level of detail and the amount of data to collect during the execution of an application, the information that can be derived in the workload characterization process may be different from trace file to trace file. If detailed trace files are used as input to MEDEA, then the tool looks for each single event entry/exit pair and, according to the event record type, correlates this new information to the previous ones in order to cumulate statistics that refer to the performance parameters used to characterize the workload components. If trace files are used that contain only statistics records, then MEDEA parses only those records that contain global information. In the following sections, specifications for the possible workload components and the corresponding parameters are given.

### 4.1 Workload components

The workload submitted to a system may be analyzed at different levels of detail, according to the "granularity" of the components selected for the modeling activity. As mentioned in §1, a workload component is defined as the basic unit of work that is considered in a quantitative description of the workload. Three different approaches (or granularities) have been adopted in MEDEA for the analysis of PICL trace files: *program–oriented*, *processor*- or *task–oriented*, and *user–event–oriented*.

In the program–oriented approach, a trace file is analyzed from a global viewpoint and information about the behavior of the application considered as a whole can be derived. The basic workload component is the program itself. The second approach derives a more detailed analysis of a trace file, in which the tasks executed on each single processor are selected as representative workload components. Finally, in the user–event–oriented approach the facility provided within PICL of defining arbitrary code segments to represent distinct workload components allows MEDEA to use the "logical" or "user" view of the application when analyzing its behavior.

### 4.2 Parallel metrics

Parallel profiles represent one of the best tools for analyzing the dynamic behavior of an application [DLP89]. If detailed PICL trace files are used as input to MEDEA, then the number of processors in use as a function of the execution time can be evaluated with respect to the different types of operations performed by the processors. MEDEA allows the user to derive execution, computation, communication, receive, and transmit profiles.

When the same application is executed several times, varying the number of allocated processors, additional parallel metrics, such as speedup, efficiency, efficacy, and execution signature, can be use to characterize the behavior of the workload [EZL89]. Note that one trace file has to refer to a single processor execution of the application itself, according to the definition of speedup, but that relative speed-up can be used when the application is too large (or too slow) to take measurements on a single processor.

### 4.3 Performance parameters

The selection of meaningful parameters to be considered in the workload characterization phase represents one of the most critical steps of this process. Table 2 lists the parameters that are currently used to characterize the program–oriented and the processor–oriented approaches.

| **Time parameters** | | | |
|---|---|---|---|
| Execution time | (extime) | I/O time | (iotime) |
| Computation time | (cptime) | Communication enable/disable time | (iptime) |
| Communication time | (cmtime) | Synchronization time | (cktime) |
| Receive time | (rctime) | Resource allocation time | (rstime) |
| Transmit time | (trtime) | System time | (sytime) |
| **Volume parameters** | | | |
| Volume of data exchanged | (ttdata) | Volume of transmitted data | (trdata) |
| Volume of received data | (rcdata) | Volume of I/O data | (iodata) |
| **Occurrence parameters** | | | |
| Number of receive requests | (rcnum) | Number of I/O requests | (ionum) |
| Number of transmit requests | (rcnum) | Number of processors | (prnum) |

Table 2: Parameters for the program–oriented and the processor-oriented approaches.

Note that these parameters may be meaningful only with respect to a particular “granularity” of the workload components. For example, the number of processors allocated to an application (parameter *prnum*) makes no sense if applied to a processor-oriented approach, where we are interested in the behavior of each single task. Furthermore, according to the workload component selected, we can have different interpretations for the same parameter. As an example, consider the case of the computation time (parameter *cptime*). If the single task has been selected to be the workload component, then the value of this parameter can be calculated as the sum of all the time intervals during which a computation is performed. This time represents the difference between the total time of the application on the specific processor and the total time spent by this processor while executing noncomputational instructions (e.g., send/receive requests, synchronization commands, etc.). Alternatively, in the case of the program–oriented approach, this parameter can be defined only as a mean with respect to the number of processors.

Table 3 lists the parameters currently used within MEDEA to characterize the user–event–oriented approach.

| **Time parameters** | | | |
|---|---|---|---|
| Total event time | (ctime) | User events time | (utime) |
| System events time | (stime) | Hidden system events time | (hstime) |
| **Occurrence parameters** | | | |
| Number of event occurrences | (cnum) | Number of hidden system events | (hsnum) |
| Number of system events | (snum) | Number of hidden user events | (hunum) |
| Number of user events | (unum) | | |

Table 3: Parameters for the user–event–oriented approach.

These parameters are completely different from those adopted for the other two approaches. In the following discussion, we use the trace records in Tab. 4 to explain meaning and usage of the parameters. Here, the first field in each record denotes an event entry (`-3`) or an event exit (`-4`), the second field denotes the event type id, and the third field denotes the timestamp for the record. The other fields can be ignored for the following discussion. System events have types less than -10, and user events have nonnegative types.

```
-3 0 0.000016 6 0 2 2 0 0                  (timestamp a)
   -3 -52 0.000128 6 0 1 2 0               (timestamp b)
   -4 -52 0.000516 6 0 3 2 8 0 0           (timestamp c)
   -3 1 0.000711 6 0 2 2 0 0               (timestamp d)
      -3 -52 0.000818 6 0 1 2 1            (timestamp e)
      -4 -52 0.001643 6 0 3 2 8 1 5        (timestamp f)
      -3 -21 0.001665 6 0 3 2 8 1 7        (timestamp g)
      -4 -21 0.001711 6 0 0                (timestamp h)
      -3 2 0.001982 6 0 0                  (timestamp i)
      -4 2 0.002005 6 0 0                  (timestamp j)
   -4 1 0.002013 6 0 0                     (timestamp k)
-4 0 0.002067 6 0 0                        (timestamp l)
```

Table 4: Example trace records.

In PICL applications, user–defined events can correspond to any arbitrary code segment. As a consequence, the presence of nested user events is very common, especially if the user events are associated with the execution of program subroutines. With respect to the example trace records in Table 4, two nested events (of types `1` and `2`) can be recognized within the "main" event of type `0`.

When these PICL trace records are analyzed according to the user–event–oriented approach, the following meanings and values are assigned to the identified parameters for user events of type `0`:

- *total event time* (ctime = 0.002051 secs) is the elapsed time between the entry record for a type `0` event (`timestamp a`) and the corresponding exit record (`timestamp l`) if the event type occurs once, or is the sum of the elapsed times if it occurs multiple times;
- *system events time* (stime = 0.000388 secs) is the sum of the execution times of any system events that are nested at the first level of type `0` events (a type `-52` event starting at `timestamp b`);
- *user events time* (utime = 0.001302 secs) is the time spent executing user events nested at the first level of type `0` events (one type `1` event);
- *hidden system events time* (hstime = 0.000871 secs) is the time spent to execute system events that are detected in nested user events (type `-52` and type `-21` events nested in a type `1` event);
- *number of event occurrences* (cnum = 1) is the number of times type `0` events have been executed on a given processor;
- *number of system events* (snum = 1) is the number of system events that are nested at the first level of type `0` events (a type `-52` event starting at `timestamp b`);
- *number of user events* (unum = 1) is the number of user events that are nested at the first level of type `0` user events (one type `1` event);
- *number of hidden system events* (hsnum = 2) is the number of system events occurring within nested user events (type `-52` and type `-21` events beginning at timestamps `e` and `g`, respectively, and nested within a type `1` event);
- *number of hidden user events* (hunum = 1) is the number of user events nested within user events at the first level (one type `2` event).

## 5 A Case Study

In the following, a brief sketch of a workload characterization study is given, to indicate how PICL trace data can be analyzed using MEDEA.

The application used for the study is PSTSWM, a message-passing benchmark code and parallel algorithm testbed that solves the nonlinear shallow water equations on the sphere [WoFo93]. This code models closely how CCM2, the Community Climate Model developed by the National Center for Atmospheric Research, handles the dynamical part of the primitive equations. PSTSWM was developed to compare parallel algorithms and to evaluate multiprocessor architectures for parallel implementations of CCM2.

PSTSWM uses the spectral transform method to solve the shallow water equations. During each timestep, the state variables of the problem are transformed between the physical domain, where the physical forces are calculated, and the spectral domain, where the terms of the differential equation are evaluated. The physical domain is a tensor product longitude-latitude grid. The spectral domain is the set of spectral coefficients in a spherical harmonic expansion of the state variables.

Transforming from physical coordinates to spectral coordinates involves performing a Fast Fourier transform (FFT) for each line of constant latitude, followed by integration over latitude using Gaussian quadrature, approximating the Legendre transform (LT), to obtain the spectral coefficients. The inverse transformation involves evaluating sums of spectral harmonics and inverse FFTs, algorithmically analogous to the forward transform.

Parallel algorithms are used to compute the FFTs and to compute the forward Legendre transforms. Processors are treated as a two dimensional grid, with the longitude dimension mapped onto row processors and the latitude dimension mapped onto column processors. Thus, the specified aspect ratio determines how many processors are allocated to computing the FFTs and the LTs. Many different parallel algorithms are embedded in the code, and the choice of algorithms is determined via input parameters at runtime.

In this study, variants of two parallel algorithm to compute the forward Legendre transforms are compared. Both parallel algorithms are based on (1) computing local contributions to the vector of spectral coefficients, (2) summing the “local” vectors element-wise over a logical ring of processors, and (3) broadcasting the result to the members of the ring. Both algorithms send $P-1$ (equal-sized) messages per processor to compute the global sum and $P-1$ messages to implement the broadcast, where $P$ is the number of processors in a processor column. Each message in the summation is sent to the logical right neighbor, while each message in the broadcast is sent to the logical left neighbor. The algorithms differ in when the three stages are executed. The first algorithm, *ringsum*, first computes all local contributions, then computes the global sum, and finally broadcasts the results. The second algorithm, *ringpipe*, interleaves the calculation of the local contribution with the global summation in a pipeline fashion, and interleaves the broadcast with the computation that uses the result, also in a pipeline fashion. The ringpipe algorithm allows the communication and computation to be overlapped, and requires less memory than ringsum. The question that was posed in the study is whether attempting to overlap communication with computation is cost effective on a given architecture.

To address the question, PSTSWM was executed on four different platforms: the Intel iPSC/2, iPSC/860, Touchstone DELTA, and Paragon machines. The Intel iPSC/2 and iPSC/860 systems are distributed memory, hypercube–connected parallel architectures. The processor elements are the Intel i80286/387 with 4MB of local memory and the Intel i860 with 8MB of local memory, respectively. The communication hardware, based on bit–serial channels, is the same for both the systems. The Intel Touchstone DELTA and Paragon systems are distributed memory, wormhole–routed, mesh–connected parallel architectures. The processor elements are the Intel i860, the same microprocessors used by Intel iPSC/860 system, and the Intel i860SP, respectively, both with 16MB of local memory.

## 5.1 Measurements

On each architecture, PSTSWM was executed on a logical 1x16 mesh topology, thus calculating each FFT sequentially and each LT in parallel. Multiple runs were made using both ringsum and ringpipe algorithms, with varying implementations of the algorithms, underlying communication protocols, and number of communication buffers. The following naming convention will be used to identify a given experiment:

`<application_name>.<algorithm_type>.<protocol_option>.<buffering_option>`

A guideline for the interpretation of trace file names is as follows:

**Algorithm type.** Each stage of both algorithms is characterized by sending data to one neighbor, receiving from another, and using the data to update a running sum. The following options differ in the order of these operations.

- ringpipe:
  1) type 00: calculate local contribution (calc)/sum/send/receive
  2) type 01: calc/sum/send/receive or calc/sum/receive/send
  3) type 02: calc/receive/sum/send
- ringsum:
  1) type 10: send/receive/sum
  2) type 11: send/receive/sum or receive/send/sum
  3) type 12: same as 10, but posting receive requests early
  4) type 13: same as 11, but posting receive request early

**Protocol option.** On Intel multiprocessors, PICL supports both blocking and nonblocking communication requests and both regular and forcetype[3] communication protocols.

1) type 0: blocking send – blocking receive
2) type 1: nonblocking send – blocking receive
3) type 2: blocking send – nonblocking receive
4) type 3: nonblocking send – nonblocking receive
5) type 4: blocking send – nonblocking receive with forcetypes
6) type 5: nonblocking send – nonblocking receive with forcetypes
7) type 6: blocking synchronous send/receive (for algorithms of type 01, 11, and 13)

**Buffering option.** When nonblocking receives and/or sends are used and extra buffer space is available, some of the receive requests can be posted "early" and some sends completed "late", potentially eliminating system buffer copying overhead and allowing additional communication and computation to be overlapped.

1) type 0: use no extra communication buffers
2) type x: use the maximum number of extra communication buffers

As an example, the trace file `pstswm.02.3.0` refers to the execution of PSTSWM using the ringpipe algorithm with the computational paradigm "calc/[send/receive | receive/send]", assuming the "nonblocking send – nonblocking receive" communication protocol and no extra communication buffers.

[3]The forcetype protocol assumes that a receive has been posted before a send request is made, thus allowing the elimination of some handshaking overhead, but requires that the user insure that this condition holds.

## 5.2 Preliminary analysis: performance parameters and parallel metrics

PSTSWM was executed on the parallel systems described in §5, varying the algorithm type and the protocol and buffer options. From each execution, a detailed trace file was collected by means of the tracing facilities provided by PICL. As outlined in §4.1, these trace files can be analyzed at different levels of detail, according to the granularity of the workload components selected, which, in turn, is a consequence of the objectives of the analysis. The usefulness of overlapping communications and computation can be evaluated by selecting the program–oriented approach: each trace file is analyzed by MEDEA from a global viewpoint and any subset of the performance parameters described in Tab. 2 can be used to characterize the behavior of the application.

In our study, total execution, computation, communication, receive and transmit times were selected as representative parameters. Their values were used as input to MEDEA, and the parallel metrics described in §4.2 were used to obtain a first insight into the dynamic behavior of the application runs.

As an example of the differences between the experimental runs, Fig. 2 and 3 show, respectively, the receive profiles derived from the execution on the Intel Paragon for `pstswm.02.4.x` and `pstswm.12.4.0`. The first is a ringpipe algorithm based on a "calc/receive/sum/send" execution paradigm. It uses the "blocking send – nonblocking receive with forcetypes" communication protocol options and the maximum number of extra communication buffers. This algorithm maximizes the opportunity for overlap of communication and computation phases. `pstswm.12.4.0` is a ringsum algorithm based on a "send/receive/sum" execution paradigm. It uses the same communication protocol as the ringpipe example, but without any extra communication buffers. The protocol option minimizes the overhead of interprocessor communication for any given send/receive pair, but the algorithmic options do not attempt to interleave the communication and computation or to eliminate all system buffer copying.

Figures 4 and 5 give a detailed view of the first communication phase shown in the receive profiles of these algorithms. Note that two subphases, summation and broadcast, can be easily identified for the ringpipe example (Fig. 4): each phase starts with a peak in the number of receiving processors, corresponding to the early posting of nonblocking receive requests by the single tasks, and then contains communications patterns involving a small number of processors at any one time as the explicit handshaking required when using forcetypes takes place. In the ringsum example (Fig. 5), there is only one peak when the summation/broadcast has been started, and then an almost continuous communication pattern can be identified as the messages move around the logical ring.

## 5.3 Workload characterization

According to the number of performance parameters selected in §5.2, the behavior of each experimental run is represented by a single point in a five–dimensional space. In our study, the statistical properties of this data set have been examined by means of the cluster analysis and the functional description modules of MEDEA for each different hardware platform,

### 5.3.1 Workload models

Since the behavior of a real workload is very complex and difficult to reproduce, and since the amount of information collected into trace files is, in general, difficult to manage, a model is usually required. Even though the execution of a workload is usually a deterministic phenomenon, it is often modeled as a nondeterministic one and statistical techniques are applied. As outlined in §3, MEDEA classifies processes in preparation for construction of workload models by means of the *k–means* clustering algorithm.

In our study, 56 trace files (corresponding to the execution of PSTSWM for the different implementations of the ringsum and the ringpipe algorithms and varying the underlying communication protocol and the number of communication buffers) have been analyzed for each architecture. The optimal partitions of the

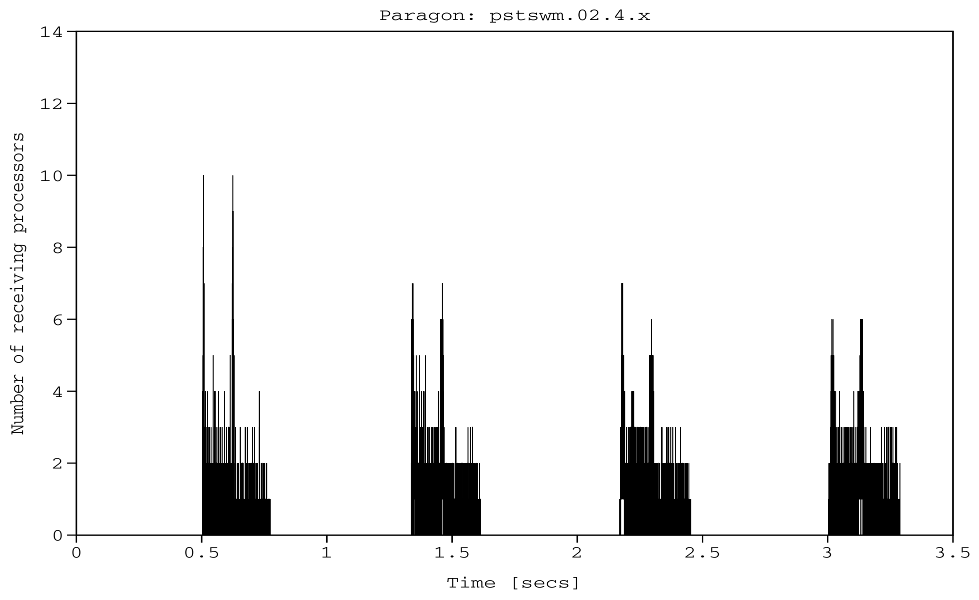

Figure 2: Receive profile for the **pstswm.02.4.x** ringpipe algorithm.

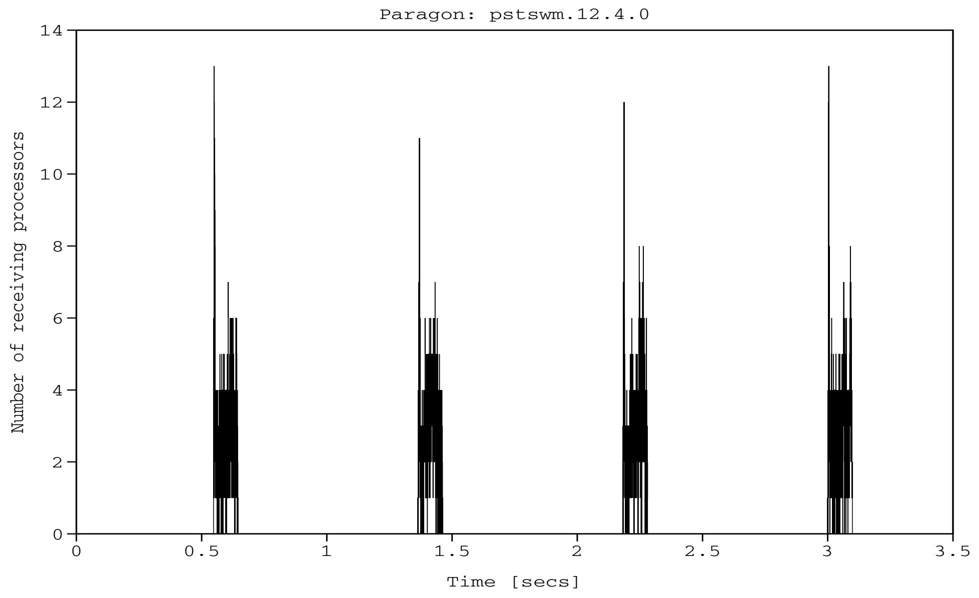

Figure 3: Receive profile for the **pstswm.12.4.0** ringsum algorithm.

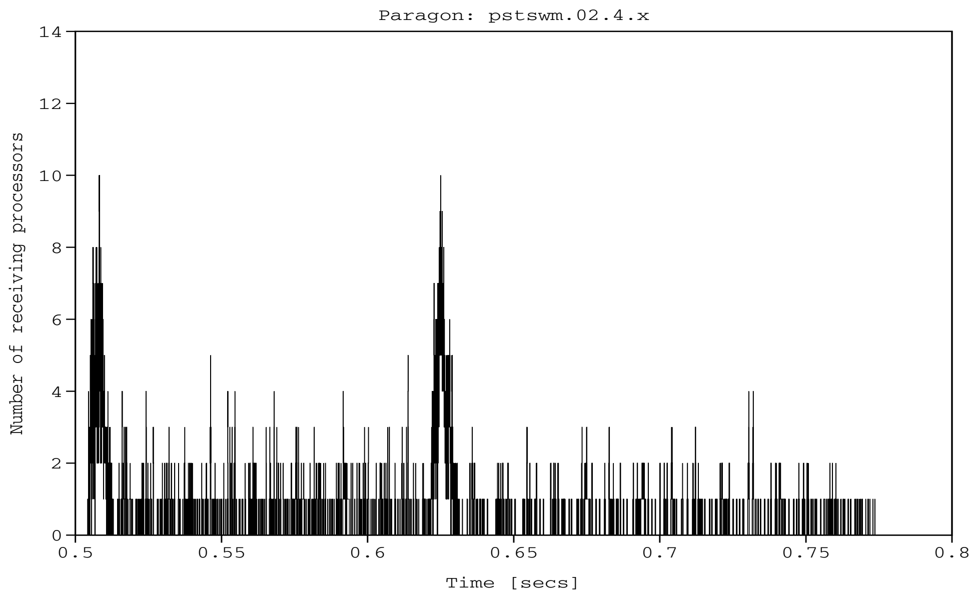

Figure 4: Detailed view of the first communication phase for the **pstswm.02.4.x** ringpipe algorithm.

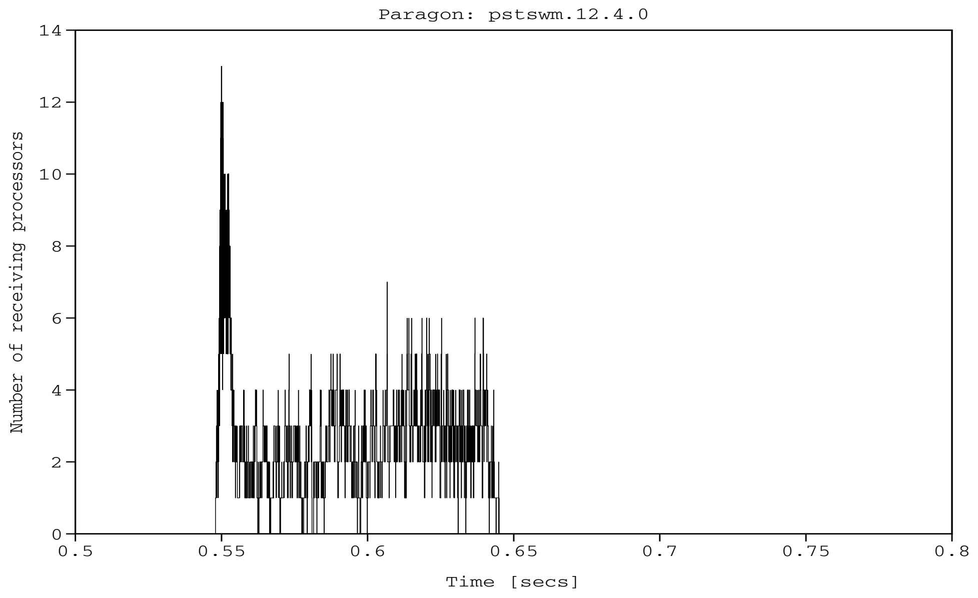

Figure 5: Detailed view of the first communication phase for the **pstswm.12.4.0** ringsum algorithm.

workload components with respect to the overall mean square ratios of the evaluated clusters are summarized in the following tables.

| | | **Extime** | | **Cptime** | | **Cmtime** | |
|---|---|---|---|---|---|---|---|
| **Cluster** | **Percentage** | mean | std dev | mean | std dev | mean | std dev |
| Cluster 1 | 17.0% | 139.577 | 0.561 | 137.659 | 0.225 | 1.712 | 0.797 |
| Cluster 2 | 47.2% | 144.483 | 0.350 | 141.450 | 0.169 | 3.013 | 0.334 |
| Cluster 3 | 35.8% | 142.571 | 0.247 | 137.376 | 0.167 | 5.514 | 0.303 |

Table 5: Workload model for the trace files collected on iPSC/2.

| | | **Extime** | | **Cptime** | | **Cmtime** | |
|---|---|---|---|---|---|---|---|
| **Cluster** | **Percentage** | mean | std dev | mean | std dev | mean | std dev |
| Cluster 1 | 1.8% | 16.158 | 0.000 | 12.836 | 0.000 | 3.321 | 0.000 |
| Cluster 2 | 28.6% | 19.788 | 0.355 | 13.089 | 0.218 | 6.690 | 0.380 |
| Cluster 3 | 19.6% | 17.849 | 0.840 | 13.307 | 0.324 | 4.525 | 0.706 |
| Cluster 4 | 3.6% | 15.463 | 0.686 | 13.869 | 0.159 | 1.585 | 0.846 |
| Cluster 5 | 46.4% | 19.802 | 0.481 | 12.904 | 0.099 | 6.891 | 0.467 |

Table 6: Workload model for the trace files collected on iPSC/860.

| | | **Extime** | | **Cptime** | | **Cmtime** | |
|---|---|---|---|---|---|---|---|
| **Cluster** | **Percentage** | mean | std dev | mean | std dev | mean | std dev |
| Cluster 1 | 20.2% | 5.227 | 0.147 | 3.947 | 0.069 | 1.278 | 0.139 |
| Cluster 2 | 34.8% | 5.849 | 0.169 | 4.185 | 0.140 | 1.663 | 0.136 |
| Cluster 3 | 15.7% | 5.835 | 0.126 | 4.756 | 0.181 | 1.074 | 0.191 |
| Cluster 4 | 20.2% | 5.679 | 0.139 | 4.205 | 0.152 | 1.471 | 0.140 |
| Cluster 5 | 9.0% | 5.152 | 0.159 | 3.947 | 0.042 | 1.204 | 0.129 |

Table 7: Workload model for the trace files collected on DELTA.

| | | **Extime** | | **Cptime** | | **Cmtime** | |
|---|---|---|---|---|---|---|---|
| **Cluster** | **Percentage** | mean | std dev | mean | std dev | mean | std dev |
| Cluster 1 | 49.1% | 3.372 | 0.053 | 3.028 | 0.029 | 0.344 | 0.344 |
| Cluster 2 | 33.3% | 3.866 | 0.039 | 3.033 | 0.033 | 0.833 | 0.049 |
| Cluster 3 | 8.8% | 3.920 | 0.128 | 3.035 | 0.028 | 0.885 | 0.110 |
| Cluster 4 | 8.8% | 3.345 | 0.011 | 3.167 | 0.042 | 0.178 | 0.045 |

Table 8: Workload model for the trace files collected on Paragon.

The means of the execution, computation and communications times represent the values for the centroid of the corresponding cluster and they can be used, for example, together with the standard deviations, as input to either analytic or simulation system models to reproduce the behavior of real workload.

Note that the problem input for PSTSWM is not necessarily comparable between the different machines. These experiments are part of a larger exercise in determining optimal algorithm parameters for problems that will be used on the largest configurations of each machine. To capture the right granularity when running on only 16 processors, the problem sizes were scaled. Thus, there is some difference between the different sets of experiments.

### 5.3.2 Functional description

The composition of each cluster has been also investigated from a functional viewpoint.

A preliminary characterization of the behavior of the different program types was indicated by displaying the projections of the experimental runs on a subspace identified by two of the parameters selected for the workload modeling activity. Figure 6 shows the projections of the ringsum and the ringpipe algorithms within the *extime–cptime* subspace for experiments run on the Paragon (see Table 8). With respect to this figure, while the first and second cluster can be easily identified, the remaining partitions do not have well defined shapes. This simply indicates that the extime and cptime parameters are not able by themselves to characterize the workload generated by algorithms belonging to the third and to the fourth cluster. If we consider the projections within the *rctime–trtime* subspace (see Fig. 7), the third and fourth clusters are well shaped too.

As a second step in our functional characterization process, the components belonging to a specific cluster can be listed in order to obtain better insights into the model of the workload being evaluated. As an example, Table 9 lists the applications grouped into the fourth cluster of the workload model for the Paragon.

| Paragon: cluster 4 | |
|---|---|
| `pstswm.02.2.x` | `pstswm.02.5.0` |
| `pstswm.02.3.x` | `pstswm.02.5.x` |
| `pstswm.02.4.x` | |

Table 9: Composition of the fourth cluster of the workload model for the Paragon.

Note that all the components belonging to this cluster correspond to trace files derived from ringpipe algorithms based on the "send/calc/receive" execution paradigm. Furthermore, this cluster groups together those PSTSWM runs utilizing nonblocking receive communication protocols and extra communication buffers. The cluster also includes the experiment utilizing nonblocking send – nonblocking receive communications with forcetypes and no extra communication buffers (`pstswm.02.5.0`). These results imply that the forcetype protocol does not change the fundamental behavior of this algorithm when using extra communication buffers, but that extra buffers are unnecessary (on the Paragon, using this algorithm) when forcetypes are used with nonblocking sends and receives.

### 5.3.3 Results

This case study has important implications on how these multiprocessors should be used. The preliminary results confirm that the utility of overlap varies across the platforms. Moreover, the techniques required to productively exploit overlapping communication with computation also vary between the architectures, even though their programming models are identical. For example, overlap is useful, and simple to characterize and exploit, on the iPSC/2. It is even more important for efficiency on the iPSC/860, but is more difficult to utilize effectively. Techniques maximizing the possibility of overlap have a marginal utility on the Touchstone DELTA, and it is doubtful whether overlap is the reason for the efficiency. The performance analysis on the Paragon currently changes with every operating system upgrade, but its performance characteristics, with regard to exploiting overlap, seem to lie between the those of the Touchstone DELTA and the iPSC/860.

The purpose of introducing this case study is to demonstrate how MEDEA can be utilized to analyze PICL trace data, what types of analyses are possible, and, hopefully, how useful the insights available from the analysis are. Our experiments on the Paragon also point out the utility of having portable tools like PICL and MEDEA. While the Paragon will have a full suite of performance monitors and tools in future releases of the system software, they were not available for these experiments. This is typical in the analysis of early or experimental systems. But it is important to understand the performance of such systems quickly, and we were not hindered in our case study by the lack of vendor supplied tools.

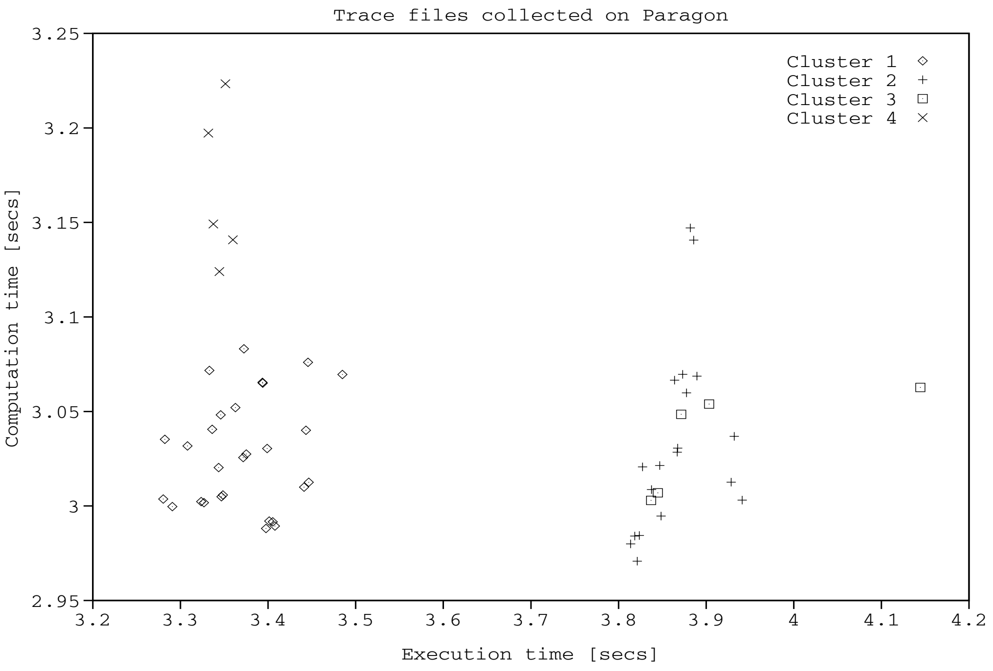

Figure 6: Projections within the *extime–cptime* subspace.

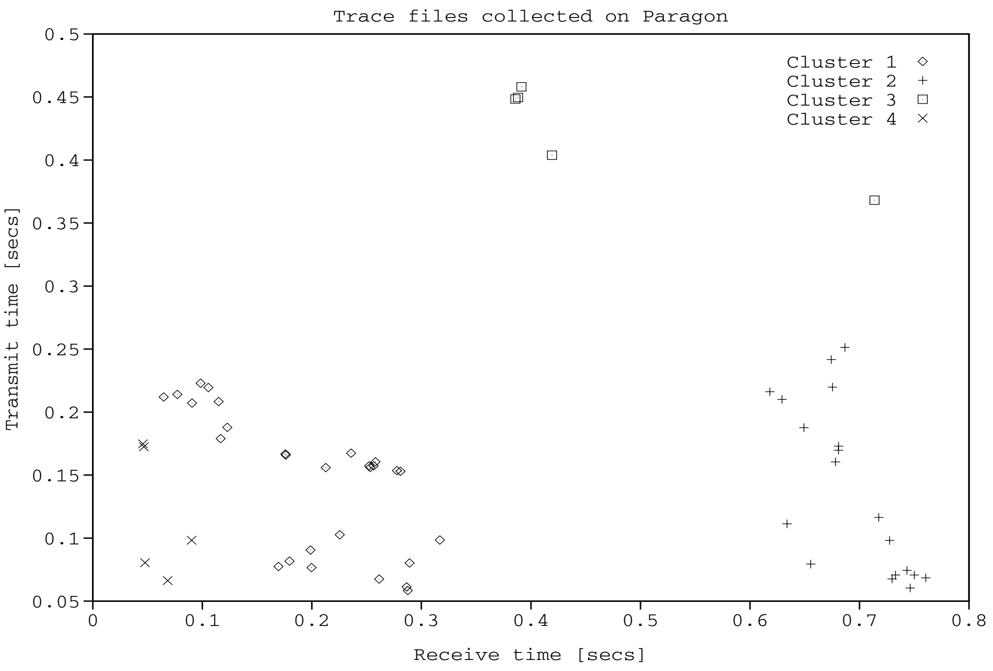

Figure 7: Projections within the *rctime–trtime* subspace.

## 6 Conclusions

In this report, the integration of the Portable Instrumented Communication Library (PICL) trace file format into the MEasurements Description Evaluation and Analysis tool (MEDEA) has been presented. This integration was motivated by the wide availability and utility of PICL trace files, and by the capabilities in MEDEA for easily analyzing the static and dynamic characteristics of parallel workloads from trace data. A workload characterization study was also presented, as a means to indicate exactly how PICL data can be analyzed using MEDEA, and to indicate what types of insight can be obtained from using these tools. In our initial experiences in using MEDEA to analyze PICL trace files, we have found the combination of these tools to be effective and powerful in workload characterization studies.

## References

[Aydt92] R. A. Aydt. The Pablo Self–Defining Data Format. Technical Report, University of Illinois at Urbana–Champaign, Urbana, IL, March 1992.

[CaSe93] M. Calzarossa and G. Serazzi. Workload Characterization: A Survey. *Proceedings of the IEEE*, 81(8):1136–1150, August 1993.

[DLP89] L. W. Dowdy, M. R. Leuze, and K. H. Park. Multiprogramming a distributed-memory multiprocessor. *Concurrency: Practice and Experience*, 1989.

[EZL89] D.L. Eager, J. Zahorjan, and E.D. Lazowska. Speedup Versus Efficiency in Parallel Systems. *IEEE Transactions on Computers*, 38(3):408–423, March 1989.

[GHPW90] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley. A User's Guide to PICL: A Portable Instrumented Communication Library. Technical Report ORNL/TM–11616, Oak Ridge National Laboratory, Oak Ridge, TN, August 1990.

[Hart75] J. A. Hartigan. *Clustering Algorithms.* John Wiley, 1975.

[HeLu91] V. Herrarte and E. Lusk. Studying parallel program behavior with upshot. Technical Report ANL/TM–91/15, Argonne National Laboratory, Argonne, IL, August 1991.

[Jain91] R. Jain. *The Art of Computer System Performance Analysis.* John Wiley & Sons, New York, 1991.

[LeSe92] P. Lenzi and G. Serazzi. PARMON: PARallel MONitor – User's Guide Release 1.0. Technical Report R3/95, University of Milan, October 1992.

[Merl93] A. Merlo. MEDEA: MEasurements Description Evaluation and Analysis tool – User's Guide Release 1.0. Technical Report R3/117, Progetto Finalizzato C.N.R. "Sistemi Informatici e Calcolo Parallelo", Aprile 1993.

[MeRo92] A. Merlo and P. Rossaro. MEDEA: Design Document. Technical Report R3/92, Progetto Finalizzato C.N.R. "Sistemi Informatici e Calcolo Parallelo", Settembre 1992.

[HeEt91b] Heath M. T. and J. A. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, (8), 1991.

[WoFo93] P. H. Worley and I. T. Foster. PSTSWM: A Parallel Algorithm Testbed and Benchmark Code for Spectral General Circulation Models. Technical Report ORNL/TM–12393, Oak Ridge National Laboratory, Oak Ridge, TN. (in preparation).