# PERFORMANCE ANALYSIS OF A PARALLEL HYDRODYNAMIC APPLICATION*

Daniele Tessera, Maria Calzarossa
Dipartimento di Informatica e Sistemistica
Università di Pavia
I-27100 Pavia, Italy
E-mail: {tessera, mcc}@gilda.unipv.it

Andrea Malagoli
Department of Astronomy
University of Chicago
Chicago, IL 60637
E-mail: malagoli@liturchi.uchicago.edu

**Keywords:** performance evaluation, monitoring, communication analysis.

## ABSTRACT

It is common experience that high performance computing applications do not fully exploit the potential parallelism of a given architecture. The reasons for such performance degradation are related to the various interactions between the design and the implementation of the application and the underlying hardware. While simple theoretical models provide useful initial estimates, they are often too general to explain the detailed behavior of a specific hardware/software configuration. In this paper we present, by means of a case study, a model for analyzing performance of parallel applications and for diagnosing sources of performance degradation. Such an approach is based on experimental measures collected at run–time. In the case study, we focus on the performance analysis of a Grand Challenge hydrodynamics application executed on an IBM SP2 system.

## 1 INTRODUCTION

There are many factors which characterize and limit the performance of an application code executed on massively parallel systems. Typically, it is not very easy for high performance computing applications, which do not consist of simple benchmark kernels, to fully exploit the potential parallelism of multi–processor systems, due to both the intrinsic applications structure and to the system architecture. For example, unbalanced data exchange between processors can give rise to chaotic and conflicting communication patterns in distributed memory systems. Hence, communications between physically remote processors can experience long delays. Conflicts on memory accesses might arise in shared memory systems. Moreover, certain parts of the application may be intrinsically sequential. All these and other situations are sources of penalties for the overall application's performance. Therefore, the intuitive idea that the performance should be roughly proportional to the number of processors allocated to an application gives a rather inaccurate estimate of what is actually achieved. Also, performance of massively parallel systems cannot be characterized by simple parameters, like MFLOPS, SPEC marks, which vendors often like to use to advertise their machines. Rather, more detailed analyses are required: different measures provide different and complementary performance estimates (see e.g., [CS93], [CMM95], [Fos95]).

Performance analysis and diagnosis are fundamental to identify those critical components within an application where tuning actions can substantially improve its overall performance. Experimental approaches, where measurements collected during the execution of the application are considered, represent the basis for such studies. Monitoring and profiling tools (see e.g., [Wor92], [RAA$^+$93], [Yan94]) are to be used for gathering at run–time the data related to the behavior of the application. A post–mortem analysis of the large amount of raw data collected is required to determine a quantitative as well as a qualitative characterization of the performance. Many tools, (e.g. [HE91], [HL91], [RAN$^+$93], [Mea94]) work for the purpose. Systematic approaches for the interpretation of the achieved performance, which is not always trivial and easy to understand and to explain, are required. These approaches have to provide compact and synthetic representations of the behavior of the application able to identify and to highlight its most relevant performance characteristics.

This paper presents on a case study a set of guidelines to be followed for performance analysis and diagnosis of parallel applications. We have chosen, as a running example, a Grand Challenge hydrodynamics code for the study of turbulent convection in stars. This code has reasonably simple structure, which makes it well suited to be used as an illustrative example. Furthermore, the numerical algorithms employed within the code are the basis for a wide range of computational fluid dynamics applications.

An extensive set of detailed performance experiments has been executed on the IBM SP2 system at Maui High Performance Computing Center. Unbalanced communication and computation activities and loss of synchronization

*The research was supported in part by the European Commission under the ESPRIT IV Project LTR n. 21033 and by the Italian C.N.R.. Malagoli was supported by the NASA HPCC Grand Challenge grant NAG5–2218 5–28537. The research was also sponsored by the Phillips Laboratory, Air Force Materiel Command, USAF, under cooperative agreement number F29601-93-2-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Phillips Laboratory or the U.S. Government.

among the processors are the main sources of inefficiencies identified in our case study.

The paper is organized as follows. Brief descriptions of the main characteristics of the application considered in our case study, of the support environment and the tools used for performance analysis are given in Section 2. The various types of analyses are presented and discussed in Section 3. Future directions in the field of performance analysis of parallel applications are outlined in Section 4.

## 2 CODE AND SUPPORT ENVIRONMENT

This section attempts to summarize the main characteristics of the application code considered in our case study with the aim of pointing out the numerical algorithms adopted together with their interactions and their parallel characteristics. The architectural features of the system where our application executed and the support environment of our experiments are also briefly described.

The code considered in our case study, developed as part of a NASA Grand Challenge project at the University of Chicago (see [MDC+95]), numerically solves the equations of compressible hydrodynamics for a gas in which the thermal conductivity changes as a function of temperature. The core of the computational algorithm consists of two independent components: a finite difference higher–order Godunov method for compressible hydrodynamics, and a Crank–Nicholson method based on nonlinear multigrid method to treat the nonlinear thermal diffusion operator. The two algorithms are combined together using a time–splitting technique.

The code, designed using an SPMD programming paradigm implemented on a message passing model, focuses on flow geometries which can be mapped on a regular rectangular grid. Therefore, we can apply standard domain decomposition techniques to distribute the data uniformly among processors. Moreover, because the finite difference operators in our scheme use only nearest neighboring points to update a grid zone, the communications tend to be quasi-local, with very few truly global communication operations being necessary.

The Godunov method used for the equations of compressible hydrodynamics is an explicit method which requires only one nearest neighboring data exchange per time step. The basic idea of the multigrid method is to increase the convergence speed of the basic relaxation iteration (e.g., a Jacobi or Gauss-Seidel iteration) by combining solutions of appropriately defined relaxation problems on several levels of coarse grids, each grid of a lower level having half the size of a grid at the upper level.

The multigrid algorithm has two different sections. The first section is similar to the Godunov scheme. The physical grid is decomposed into sub-domains, which are then mapped onto different processors. Each local domain has its own injection into the coarse grid. When the grid size per processor becomes too small, the second section begins.

Each relaxation step requires nearest neighbors data exchange, at all levels of the grid. In the second section, the nearest neighbors data exchange becomes very expensive relative to the computation at each processor. Hence, we do a global collection of all the local domains, to provide a copy of the complete grid to all the processors. We continue with the process of injecting into coarse grids until we have a small enough grid to be solved exactly. The entire process of injecting, relaxing and prolonging back is completely local to the processors, and is performed redundantly, but concurrently, on each processor. This redundancy eliminates the need for further global communications during the prolongation phase to the non local meshes.

The code has been implemented using Fortran 77 and C languages. Functions from the Chameleon library (see [GS93]), which provides a uniform interface to the communication systems available on various parallel systems, are used. This approach is the key to insure portability of the code across parallel architectures and performance improvements by tuning communication protocols.

As part of the support environment for our study, the monitoring facilities of the Chameleon library have been used to collect at run–time information related to each communication event (i.e., beginning, end, type). This information, stored into tracefiles according to the ALOG format (see [BL94]), has then been processed by means of Medea (see [CMM+95], [CMM+96]), the tool used for the evaluation and the visualization of the performance achieved by our application.

The performance of our hydrodynamics code has been tested on the IBM SP2 at Maui High Performance Computing Center. The system consists of 400 processors (nodes), grouped together into frames containing 8 or 16 processors directly linked to a switch board, which provides a fully connected topology (see [SSA+95]). Additional switch boards are used to provide interconnections between frames through a multistage network.

## 3 PERFORMANCE ANALYSIS

As already discussed, it is quite difficult to extract and to select among the large amount of information collected by monitoring tools the most relevant information able to explain the behavior and the performance of a parallel application. Various types of statistical and numerical analysis techniques need to be applied for performance evaluation and diagnosis. These analyses have to identify the inefficiencies within the code by pointing out the achieved and the achievable performance and the portions of code which influence and determine such performance.

In what follows, we apply a hierarchical approach which starts from high level information (i.e., static approach) and then goes deeper and deeper into the analysis (i.e., dynamic approach and SPMD analysis) to capture more and more detailed performance information.

This case study should provide a sort of guidelines since, as we will see, the obtained results are general enough that can be used for a large variety of application domains.

In order to keep our experiments uniform, we have used a dedicated partition of the IBM SP2 with 64 processors only. Note that, although production runs of applications dealing with interactive solvers typically require thousands of iterations, a small number of iterations (i.e., ten) is enough to analyze the behavior of the embedded solver and to predict its performance. Moreover, in order not to perturb our analyses and our predictions, we have decided to discard the first iteration since it experiences all the delays related to the initial synchronization of the allocated processors.

### 3.1 Static Approach

As a first high level description, our application has been characterized by a few static parameters, e.g., execution, communication and computation times, which give preliminary insights into its performance. In particular, the exploitation of the potential parallelism available with a given number of processors is shown. The execution time for a problem of 512×512 points distributed among 32 dedicated processors of the IBM SP2 is equal to 15.83s; the computation and communication times are 12.32s and 3.51s, respectively. The communication time accounts for about 22.17% of the overall execution time. When the percentage of communication time is small, each processor spends most of its time in computation activities. Hence, as a rule of thumb, we can say that the performance can be improved by allocating a larger number of processors. For example, by allocating 64 processors to the same problem, we experience a 30.51% decrease of the execution time, which is equal to 10.45s. On the other hand, when the percentage of communication time is dominant, each processor spends a considerable amount of time in communication activities. In this case, an increase of the number of allocated processors may even result in an increase of the overall execution time. Then, it might be worth to focus the analysis on the communication policies adopted by either the library and the system, or on the communication patterns of the application.

As part of the static approach, we have considered a set of runs of our application with varying the number of processors and the problem size. The objective of these analyses is to study the scalability of the application, that is, to show how it is able to exploit the available parallelism. Figure 1 presents the execution times for problem sizes ranging from 512×512 up to 8192×8192 when the number of allocated processors varies from 1 to 64. As can be seen, the application scales fairly well according to the number of allocated processors. However, there are a few inefficiencies which degrade the achieved performance. For example, on a problem size of 512×512 going from 32 to 64 allocated processors, we have an increase of about 4% of the time spent by each processor in communication activities even though the amount of exchanged data (i.e., 2.5Mbyte) decreases by 32% due to smaller local domains. This is due to an increase of contentions on the interconnection network. Another source of performance degradation is the computation related to the extra points required by the finite difference operators, i.e., their stencil. When the number of allocated processors is doubled, since the size of the local domains is halved, the computation time for each processor

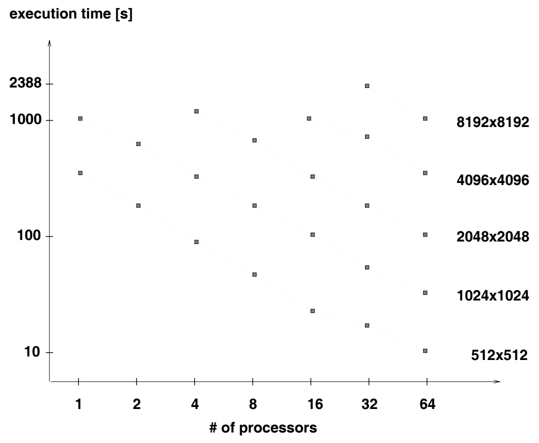


Figure 1: Execution times for problem sizes ranging from 512×512 up to 8192×8192 as a function of the number of allocated processors.

should be halved. In our case, this reduction is equal to 45% only, due to stencil computation. Hence, these extra computations may be not negligible and have to be taken into account when dealing with scalability analysis.

As a part of the static approach and as a preliminary study towards the dynamic analysis, the communication activities have been characterized by their type and by the amount of exchanged data. In what follows, we focus our analyses on a run of a problem size of 512×512 points on 64 processors.

Most of the communications (99.49%) are point-to-point `send/receive` operations due to the exchange of boundaries of local domains. There are very few global operations involving all the allocated processors. The communications are also characterized by an exchange of small messages whose average length is equal to 1051 bytes. Figure 2 presents the distribution of the length of the 446,336 point-to-point messages exchanged by the 64 allocated processors. The 70-th percentile of the distribution (equal to 296 bytes) is also shown. Such a distribution provides useful information on the communication requirements of the application and drives the dynamic analysis.

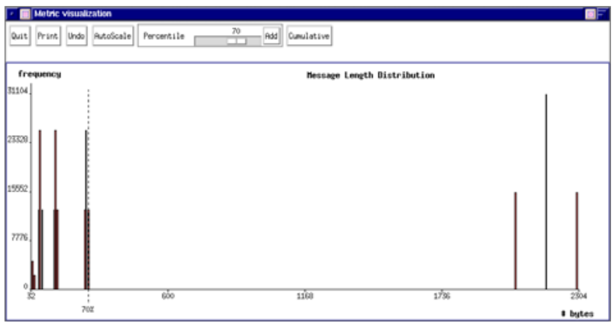
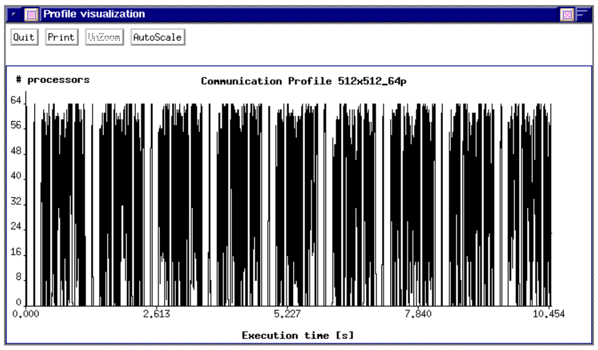
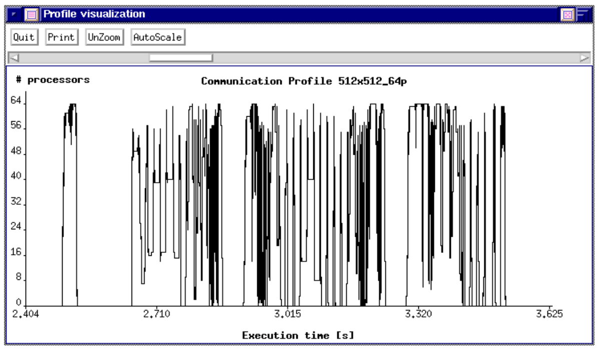


Figure 2: Distribution of the length of the point-to-point messages exchanged by the 64 allocated processors.



(a)



(b)

Figure 3: Communication profile (a) and a zoom over one phase (b) for a run with 64 processors.

## 3.2 Dynamic Approach

The dynamic behavior of the communication activities has been studied by analyzing the profile curves which represent, as a function of the execution time, the number of processors involved in simultaneous communication activities. The area below the curve represents the time spent by all the processors in communication activities.

Figure 3(a) shows the communication profile for our application, where the 512×512 grid has been equally distributed among the 64 allocated processors. In the case of perfectly balanced work and data distributions, data parallel applications should exhibit very sharp profile curves, i.e., all the processors should start and complete the same activity at the same instant of time. However, this does not occur in general and neither in our case, as can be seen from Fig. 3(b), which is a zoom over one of the nine iteration steps shown in Fig. 3(a). The processors are not exactly synchronized as expected. Indeed, by looking at the first communication phase, i.e., the first peak of Fig. 3(b), we notice that although almost all the processors (about 60) start simultaneously their communications, they have to wait for the very few remaining (just 3 or 4) in order to complete the communication. As a consequence, the delays of few processors are experienced by all the others with a consequent degradation of the performance due to an increase in the overall communication time.
The situation of processors waiting for few others, as shown by communication profiles, may denote the presence either of unbalanced data or work distributions within the application and of inefficiencies in the exploitation of the available parallelism. Hence, the analyses have to be focused on more specific activities, i.e., `send`, `receive`. The transmitting and receiving profiles are shown in Fig. 4(a) and Fig. 4(b), respectively. The two profiles represent the number of processors simultaneously sending or receiving messages as a function of the execution time. Although the same blocking communication protocol is adopted for both `send` and `receive` activities, the shapes of the two profiles are very different. A `send` is completed when a message reaches the system buffer of the source processor, without implying that the message itself has been received by the destination processor. Hence, transmissions are much faster than `receive` activities because they only require the availability of free buffers. Indeed, the IBM SP2 implementation of this protocol forces the completion of the data transfer to/from the application and system buffers before returning the execution to the application itself.

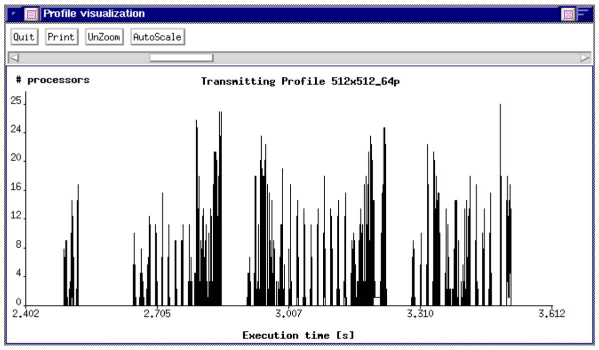
The transmitting profile of Fig. 4(a) is characterized by sharp changes in the number of processors simultaneously performing a `send` operation. The figure also shows that the number of processors whose activities are well synchronized ranges from 10 up to 25. This means that there are fluctuations in the synchronization among the allocated processors. Furthermore, because each operation requires, on the average, only 46$\mu$s (with a standard deviation equal to 20$\mu$s), we never find more than 25 processors (out of the 64 allocated) simultaneously sending messages.

The receiving profile (see Fig. 4(b)) is smoother than its transmitting counterpart; it presents larger peaks involving from 50 up to 60 processors, waiting for the completion of `receive` operations. This behavior is mainly due to transmission delays, since the time spent to transfer data from the communication to the application buffers is equal to the time to transfer data from the application to the communication buffers. The contentions among the processors to obtain free communication links on the interconnection network are responsible for such delays. As we will explain in more details later on, this is due to the contentions on the interconnection network of the IBM SP2.

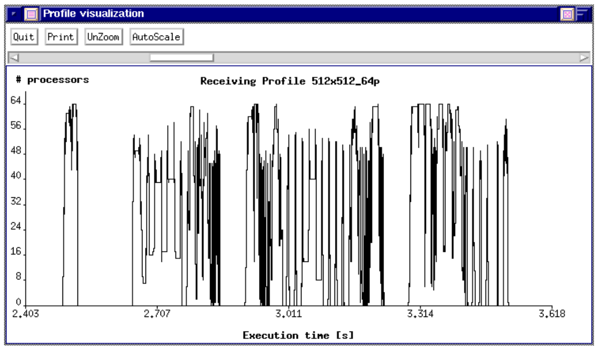
## 3.3 SPMD Analysis

As part of our hierarchical approach towards performance analysis, we have applied the SPMD analysis which focuses on the individual communication activities issued by each allocated processor. This approach has been named after the SPMD (or data parallel) programming paradigm. As already pointed out, because of this paradigm, all the processors issue the same operations. After the computations on its local domain, every processor has to exchange the boundaries with its neighbors. This implies that all processors start a `send` operation followed by a `receive`.

The goal of the SPMD analysis is to investigate the similarities of the behavior of the processors. Clustering techniques, implemented within Medea, have been used for the purpose; the behavior of each processor has been de-



(a)



(b)

Figure 4: Number of processors, as a function of execution time, sending (*a*) and receiving (*b*) messages during one iteration step.

scribed by five parameters corresponding to the times spent for communication, transmission, receiving, for performing global operations, that is, for extrema findings and for collecting data. Clustering techniques, applied to these 64 tuples, yield a subdivision of the processors in two groups. The first group contains 34 of the 64 allocated processors which are characterized by smaller times; in particular, these processors are 11% faster in collecting data than the remaining 30 belonging to the second group. The mean times spent in these operations are equal to 0.152s and 0.171s, respectively. Let us recall that the 64 processors allocated to our application are hardwired eight by eight to eight different frames. The cluster subdivision outlined that all the processors but two, belonging to the same frame exhibit a similar behavior. This result confirms the idea that processors within the same frame reach a better synchronization. Contentions arising when accessing shared links in the multistage network are responsible of losses of synchronization.

As a further refinement, the behavior of the individual processors has been analyzed by applying clustering to the 7010 communications required by the application on each processor. Two groups of communication activities have been identified. The first group which accounts for 87.8% of the communications, consists only of `send` (56.7%) and of `receive` (43.3%) operations. The second group, in turn, is composed by the remaining 855 communications subdivided in `receive` (97%), global collection (1.5%) and global minimum (1.5%). A synthetic description of these groups is given by their centroids, i.e., their geometric centers, shown in Fig. 5. The centroids represent the mean times spent to perform a communication activity on a specific processor. The gray and the black bars refer to the first and the second group, respectively. As can be seen, communications belonging to the first group require, on the average, times ranging from 0.158ms (processor 36) up to 0.218ms (processor 43), while the mean times of communication activities belonging to the second group vary from 2.677ms (processor 43) up to 3.182ms (processor 21). Moreover, note that the communication library rearranges the boundary exchanges of local domains in order to minimize the contentions on the communication network and hence, due to both these contentions and the communication policies, individual processors exhibit different behavior. For example, processor 36 is very fast in performing a few communication activities (which belong to the first group) and it is very slow for few others (which belong to the second group).

The information obtained by this approach, i.e., by focusing on the communication activities issued by each processor, can be very useful for evaluating the effectiveness of tuning actions on the communication policies. Per protocol analyses provide good evaluations on the performance improvements achievable by overlapping, for example, `send` and `receive` activities at the expenses of extra buffers. As a result of this study, we have obtained useful information which are the basis for tuning and optimizing the performance of our code.

## 4 CONCLUSIONS

We have explored a detailed evaluation of the achieved (or achievable) parallel performance of a real–life high performance computing application code. In order to present a working example, we have focused on the case study of a specific application code which is part of the NASA HPCC Grand Challenge program. The objective of our analyses is the identification of the portions of the code sources of inefficiencies which are responsible of performance degradation. Clearly, the combined roles of the parallel architecture, its operating system and the application software are to be considered together and cannot be evaluated separately. For this reason, an experimental approach, whereby performance is measured, is likely to provide useful information that can lead to performance improvements, while providing data useful to the construction of more sophisticated theoretical performance models.

We have presented and discussed a number of performance indicators that can be derived from the measurements collected by monitoring tools. We have also discussed how to use and to interpret the information provided by these indicators and how to relate the observed performance (or lack of thereof) to the characteristics of the application code and of the architecture.

Future work will be dedicated to test this methodology on different applications (e.g., distributed Fast Fourier

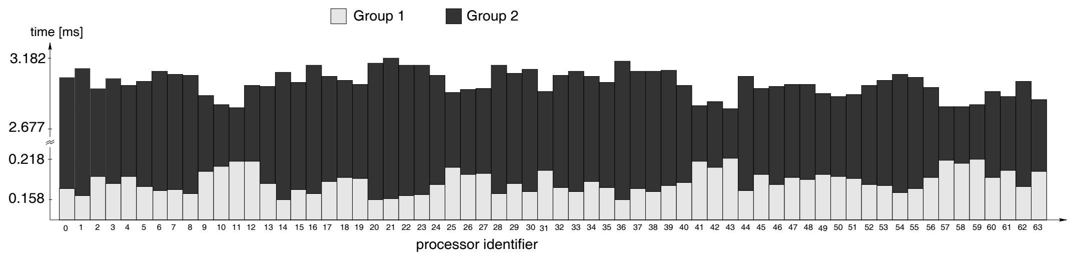


Figure 5: Mean time, for each of the two groups, spent to perform communication activities on each processor.

Transforms and adaptive mesh refinement) and to derive theoretical models of the performance based on experimental approaches.

## References

[BL94] R. Butler and W. Lusk. Monitors, Messages, and Clusters: The p4 Parallel Programming System. *Parallel Computing*, 20:547–564, 1994.

[CMM95] M. Calzarossa, L. Massari, and A. Merlo. Performance Analysis of Concurrent Software: issues, methodologies and tools. In G. Balbo and M. Vanneschi, editors, *General Purpose Parallel Computers: Architectures, Programming Environments, and Tools*, pages 287–298. ETS, 1995.

[CMM+95] M. Calzarossa, L. Massari, A. Merlo, M. Pantano, and D. Tessera. MEDEA – A Tool for Workload Characterization of Parallel Systems. *IEEE Parallel and Distributed Technology*, 2(4):72–80, 1995.

[CMM+96] M. Calzarossa, L. Massari, A. Merlo, and D. Tessera. Parallel Performance Evaluation: the MEDEA Tool. In H. Lidell, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *High-Performance Computing and Networking*, volume 1067 of *Lecture Notes in Computer Science*, pages 522–529. Springer–Verlag, 1996.

[CS93] M. Calzarossa and G. Serazzi. Workload Characterization: a Survey. *Proc. of the IEEE*, 81(8):1136–1150, 1993.

[Fos95] I.T. Foster. *Designing and Building Parallel Programs.* Addison–Wesley, 1995.

[GS93] W. Gropp and B. Smith. Users Manual for the Chameleon Parallel Programming Tools. Technical Report ANL-93/23, Argonne National Laboratory, 1993.

[HE91] M.T. Heath and J.A. Etheridge. Visualizing the Performance of Parallel Programs. *IEEE Software*, 8:29–39, 1991.

[HL91] V. Herrarte and E. Lusk. Studying Parallel Program Behavior with `upshot`. Technical Report ANL-91/15, Argonne National Laboratory, 1991.

[MDC+95] A. Malagoli, A. Dubey, F. Cattaneo, and D. Levine. A Portable and Efficient Parallel Algorithm for Astrophysical Fluid Dynamics. In *Parallel Computational Fluid Dynamics*, pages 553–560. North–Holland, 1995.

[Mea94] A.D. Malony et al. Performance Analysis of pC++: A Portable Data-Parallel Programming System for Scalable Parallel Computers. In *Proc. Eighth Int'l Parallel Processing Symp.*, pages 75–84. IEEE Computer Society, 1994.

[RAA+93] B. Ries, R. Anderson, W. Auld, D. Breazeal, K. Callaghan, E. Richards, and Smith W. The Paragon performance monitor environment. In *Proceedings Supercomputing '93*, pages 850–859. IEEE Computer Society, 1993.

[RAN+93] D.A. Reed, R.J. Aydt, R.J. Noe, K.A. Shields, B.W. Schwartz, and L.F. Tavera. Scalable Performance Analysis: The Pablo Performance Analysis Environment. In *Proceedings Scalable Parallel Libraries Conference*, pages 104–113. IEEE Computer Society, 1993.

[SSA+95] C.B. Stunkel, D.G. Shea, B. Abali, M.G. Atkins et al. The SP2 High-Performance Switch. *IBM Systems Journal*, 34(2):185–204, 1995.

[Wor92] P.H. Worley. A New PICL Trace File Format. Technical Report ORNL/TM–12125, Oak Ridge National Laboratory, 1992.

[Yan94] J.C. Yan. Performance Tuning with AIMS - An Automated Instrumentation and Monitoring System for Multicomputers. In *Proceedings of the 27th Hawaii International Conference on System Sciences*, volume II, pages 625–633, 1994.

"