Clover: Efficient Monitoring of HPC Clusters

D. Montaldo, E. Mocskos, D. Fernández Slezak

Laboratorio de Sistemas Complejos, Departamento de Computación,
Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires
Buenos Aires (C1428EGA), Argentina

Abstract. As a consequence of the last decade evolution of computational power and the proliferation of clusters, i.e. collection of interconnected personal computers, system monitoring has become a critical and non-trivial task. Specialized protocols have been developed to provide cluster state information without degradation of performance and avoiding the consumption of computational power.
ClOver is a system monitoring tool designed on a plugin architecture based on the CluMon project, a tool developed at the National Center for Supercomputing Applications (NCSA). The main goal is to allow the verification of the cluster state at a glance. It includes a plugin implemented over the Intelligent Platform Management Interface (IPMI) protocol to collect data with almost no CPU-cycle consumption.
The aim of this paper is to show the *ClOver* current state of development. Furthermore, the benefits of using IPMI in monitoring activities are verified running the simulation of the heat equation as a test application in both shared and distributed memory architectures.
The results obtained suggest that the use of specialized hardware protocols for sensoring and monitoring would save valuable CPU cycles.

1 Introduction

Over the last few years the computational world has witnessed an unprecedented progress in information technology. The advance in microprocessors, storage capacity, networking speed and telecommunications in general has provided the underlying infrastructure needed to build a powerful computational structure. Such structure is designed to satisfy the increasing demands of computing power required by scientific and commercial applications.

A consequence of the evolution of computational power and storage capacity during the last decade is the proliferation of *Cluster Computing*. Clusters are a collection of interconnected personal computers that are used as dedicated resources for the resolution of problems (parallel processing, task farming, etc.). Because of their simple structure, scalability and low cost, clusters have become the favorite choice for scientific environments. Nowadays, more than 50% of the 500 fastest computers in the world are clusters[1].

Due to system growth, big-scale cluster monitoring has become a non-trivial task. The amount of information generated by monitoring systems requires specialized protocols and software to recollect, process and show this information.

As a result, protocols capable of providing this information in real-time without degradation of performance or consumption of computational power have arisen. Intelligent Platform Management Interface (IPMI)[2] emerges as the standard of these specialized protocols.

In this article, *ClOver* (Cluster Overseer) is presented: a new platform for cluster monitoring, with efficient resources use and designed for extension using a flexible plugin architecture. *ClOver* includes a new plugin for low consumption of CPU-cycles, which is based on the IPMI protocol for sensor information collection.

The advantages and disadvantages of the most popular monitoring systems are show in section 2. In section 3 the design details of *ClOver* are presented. Finally, the results obtained using a test application in a small-scale cluster are exposed in section 4 and some conclusions are given in section 5.

2 Related Work

Since the construction of the first clusters, monitoring their state to detect failures and keeping status history has been a basic task. In this sense, many tools have been developed to assist the cluster administrator for this purpose.

Following, some of the best-known applications developed are presented.

2.1 Supermon

Supermon[3] is one of the first monitoring tools developed. Its main goal is keeping low perturbation of systems, high refresh rate and a flexible interface and communication protocol. It may scale up from one processor up to cluster over 1024 processors. Moreover, the communication protocol is open, so it would be feasible to implement clients on many platforms.

However, this software lacks a web interface necessary for modern cluster administration and, as a design principle, recovers sensor information using only software components, adding perturbation to the system. Developing this interface and enabling hardware sensoring would require an expensive re-engineering process.

2.2 Hawkeye

Condor[4] is a specialized system in job management for high-throughput tasks. As well as other batch-type systems, Condor provides a queue mechanism, scheduling policies, priority schemes and resource administration and monitoring. Users may send standard or parallel jobs to Condor, which are queued. Resources are selected based on the active scheduling policy, and jobs are monitored until their execution is finished, when users are informed.

Hawkeye[5] is the Condor component in charge of resource monitoring. For its correct functioning, it needs Condor running in order to get periodic execution of specific jobs, generally scripts. Most of the jobs produce an output with tuples

(attribute/value) describing the current resources state. This information is then indexed and published for system access.

As can be seen, Hawkeye is extremely coupled with Condor, and making use of it independently is almost impossible.

2.3 Nagios

Nagios[6] is a system and network monitoring application. It monitors specified hosts and services, alerting when things go wrong and when they improve.

Nagios' architecture is strongly based on a plug-in mechanism. Nagios itself knows nothing about the details of the monitoring information; instead, it relies on plugins to do all the work.

Plugins act as an abstraction layer between the monitoring logic present in the Nagios daemon and the actual services and hosts that are being monitored.

Currently, there are plugins available to monitor many different kinds of devices and services, including: HTTP, POP3, IMAP, FTP, SSH, DHCP, CPU Load, Disk Usage, Memory Usage, Current Users, Unix/Linux, Windows, and Netware Servers, Routers and Switches and others.

The status of the nodes being monitored can be checked through a web based interface, including different views to help the administrators find the problems in a quick and easy way.

One of the main advantages of Nagios is, at the same time, one of the drawbacks for using it in the case of clusters: it can monitor almost anything, but because of this feature it lacks focus on specific hardware architecture, and so it can not provide a unified vision of the cluster status at a glance. This is a key feature for cluster administration, when there is a need to check the status of thousands of nodes and this *one-look check* becomes very valuable.

2.4 Ganglia

Ganglia [7, 8] is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization.

Nowadays, Ganglia is one of the most widely used tools for cluster monitoring. It keeps very active and is extensively supported in the community. Nevertheless, the interface is oriented to show a great amount of information for each node in a very fancy way, which prevents users and administrators from having a summarized view of the whole cluster at a glance, specially in the case of big-scale clusters.

2.5 CluMon

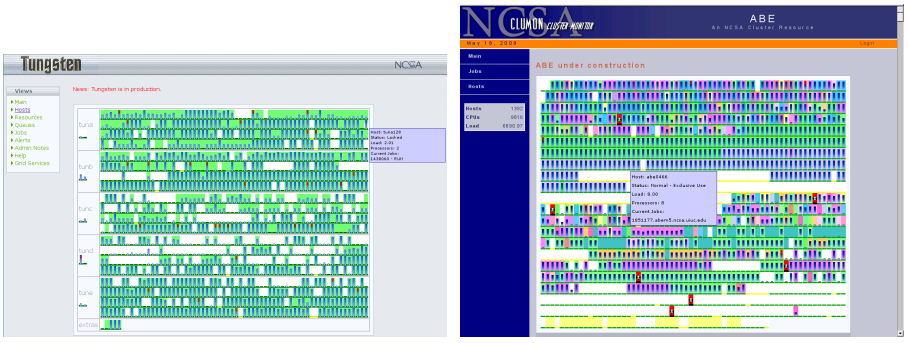
The CluMon [9] performance monitoring system was developed for monitoring Linux-based clusters at the NCSA at the University of Illinois, Champaign-Urbana. The system is currently based on Performance Co-Pilot (PCP) by SGI

and the Portable Batch System (PBS) scheduler. It also uses MySQL as its database, and Apache with PHP as its bundled viewer. The system was created to provide an overview of what the current state of the cluster at a glance.

PCP[10] provides services for monitoring systems, specialized in performance. For example, tracing and analyzing temporary lost of performance, in order to correlate quality level of users with activity on nodes, or diagnose complex situations with resources demand. To obtain information, PCP runs a system-level daemon that is executed when called. This architecture makes it simple to implement a centralized monitoring system, specially for big-scale clusters.

CluMon presents a good solution for monitoring big-scale clusters because of the use of PCP and the specialized Graphical User Interface (GUI). Working examples of this web interface are presented in Fig. 1, which includes two big-scale clusters with more than 1200 nodes. Moreover, through its connection to PBS, CluMon may expose jobs state in the cluster.

All the information fits perfectly in one screenshot. The usage of small graphs with colors indicating the node state provides the capacity of easily identifying the problematic nodes. Additional information can be obtained by pointing the mouse over each node.



(a) clumon-w.ncsa.uiuc.edu (b) abemon.ncsa.uiuc.edu

Fig. 1. Two different clusters running the CluMon web interface. The cluster in figure (a) has 1300 nodes, while the other one has 1200. It is very easy to check the status of the systems. Moreover, pointing the mouse over the nodes provokes a pop-up summary of their state to appear.

This project includes the main *ClOver* requirement, i.e. monitoring of big-scale clusters at a glance. However, CluMon does not provide enough flexibility in the plugin's design and is very coupled with PCP. Even though CluMon claims new plugins can be added, it does not support removing the PCP plugin. Moreover, the PCP plugin does not allow the selection of which sensors to measure, making impossible to configure the system to capture some sensors from PCP and some others from other programs or protocols. On the other hand, the

current plugin design in CluMon lacks of an interface for specific configuration pages of plugins, accessible by the user in the GUI.

Based on the previous considerations, CluMon was selected as the base project for the new resource monitoring tool. In this new tool, *ClOver*, plugins template were re-engineered and the functions that show information were modularized.

3 *ClOver* (Cluster Overseer)

ClOver is a resource monitoring system based on CluMon, with the same high-level architecture, shown in Fig. 2. The main engine was re-engineered and re-implemented completely; the plugins scheme was designed to allow simple addition of new features following the main ideas of Nagios. Currently, information of each sensor may be collected from PCP or IPMI, but *ClOver* can be easily extended.

IPMI[2] defines a standardized, abstracted, message-based interface to intelligent platform management hardware and standardized records for describing platform management devices and their characteristics. Among others, it provides the capability of monitoring server hardware sensors, i.e. system temperature, voltage, fans, power supplies, bus errors and system physical security; centralized and non-volatile system event log; and remote console applications to provide access to text-based interfaces for BIOS, utilities, operating systems, and applications while simultaneously providing access to IPMI platform management functions. Moreover, platform management functions can also be made available when the system is powered off (but plugged in).

Introduced in 1998 by Intel, HP, NEC and Dell, IPMI defines a standard set of messages for the characteristics of hardware components. Sensors funnel data into a Baseboard Management Controller (BMC), which sends IPMI messages over the server's system bus to the network as well as over an independent Intelligent Platform Management Bus (IPMB). Sensors data may be collected without stealing CPU-cycles.

Figure 2 displays a cluster with nodes connected through a Local Area Network (LAN), with one node for monitoring purposes. This cluster may be composed of IPMI-enabled nodes or nodes with only PCP installed. Collected information (using IPMI and/or PCP) is then processed and saved in a MySQL database. Finally *ClOver* publishes the results in a web server using PHP allowing the access to information from any web browser.

3.1 Plugins

The *ClOver* plugin scheme was designed following the CluMon management principles and Nagios architecture ideas. Execution and management of plugins are done by Portable Operating System Interface for Unix (POSIX) process primitives such as `fork` and `join` (or `wait`), and by handling the signals to control them. This powerfull mechanism provides a detailed control over the

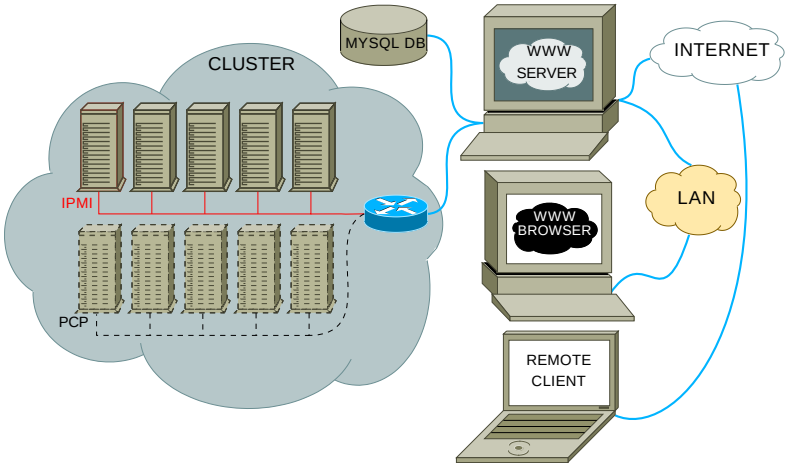


Fig. 2. *ClOver* component diagram and scheme of internal interaction. The cluster have both IPMI-enabled and PCP nodes.

collecting processes using signals and handlers like `SIGTERM`, `SIGUSR1`, `SIGUSR2`, etc. Following Nagios architecture, the *ClOver* engine knows almost nothing about the details of the monitoring information, relying on plugins to do all the work.

In each collection cycle, the enabled hosts are harvested, and for each host the enabled plugins are forked with the corresponding parameters.

Once the last fork of each node has been launched, the main engine waits for plugins to finish. If a predefined timeout is reached, a termination signal is sent (`kill`) to the executing plugins.

Figs. 3 and 4 present two examples of the plugins control mechanism. In both cases, the main engine starts two child processes. In Fig. 3, execution is completed without reaching timeout. On the other hand, in Fig. 4, the timeout period elapsed prior to the completion of both child processes. Moreover, the timeout occured because of different reasons in each child: the first one received data from the host but was not capable of processing it on time, while the second one did not receive the information from the node.

4 Results

The main goal of this section is to demonstrate that IPMI consumes less compute resources than PCP to collect information. In order to analyze this, collection of data using both plugins are done while CPU-intensive jobs are run.

Two test-bed scenarios are proposed to evaluate the behavior of these plugins on different architectures: shared and distributed memory. In each scenario, three different conditions are evaluated:

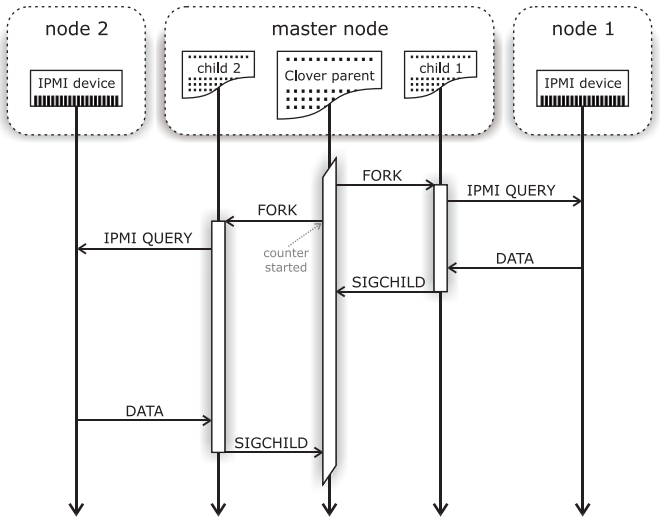


Fig. 3. Interaction diagrams between one monitoring node and two compute nodes. In this scenario, both child processes get the information and send it to the mail engine on time, not reaching the timeout.

Control application run without any monitoring interference.
PCP calculation done with PCP plugin enabled for system monitoring.
IPMI calculation done with IPMI plugin enabled for system monitoring.

In order to avoid interferences caused by operating system services, only the strictly necessary ones are enabled. The elapsed time between application invocation and termination (hereinafter *wall time*) is used to show the impact of CPU consumption in each protocol.

4.1 Shared Memory Architecture

In this type of architecture, running processes share memory spaces, allowing interprocess communication using variables in common memory space and providing fast access to them. The applications designed for this architecture use multithread programming techniques, being OpenMP[11, 12, 13] one of the most widely used Application Programming Interface (API) supporting this.

The hardware used is based on an Intel Server Board S5000PSLSATA, with two Intel Dual Core Xeon 5030 processors (2.67 GHz, 4MB L2 cache), 4GB of RAM (in two banks) and 80GB SATA hard disk (8MB cache).

The application chosen is the calculation of heat distribution in a one dimensional pipe using OpenMP API obtained from Berna L. Massingill's webpage[1].

[1] It can be obtained from http://www.cs.trinity.edu/~bmassing/Classes/CS3366_2009spring/SamplePrograms/OpenMP/heat-eqn-par-spmd.c

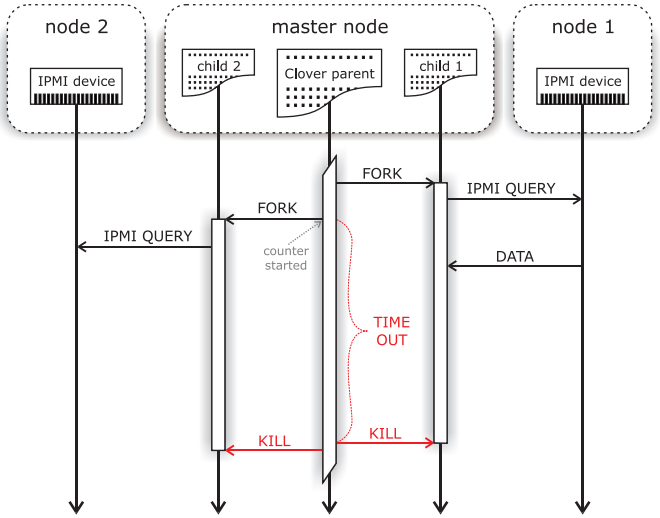


Fig. 4. Interaction diagram between one monitoring node and two compute nodes. In this scenario, both child processes reach the timeout, so the `kill` signal is sent.

In Fig. 5, results of running this application with four threads are shown. As was previously stated, three columns are plotted in the figure including the standard error in each one. The IPMI and control times are almost the same, even though they are statistically different. The PCP running time is considerably larger than the other two cases, clearly showing the benefits of using IPMI protocol.

4.2 Distributed Memory Architecture

For distributed memory architecture, the most common infrastructure is the Beowulf-type cluster [14]. This consists of off-the-shelf commodity computers connected through standard ethernet LAN[15]. The Message Passing Interface (MPI)[16] is the de facto standard nowadays to program distributed memory applications, it gives a set of primitives to communicate processes using messages. There are several MPI implementations including open source and closed commercial ones, being OpenMPI [17] and MPICH[18, 19, 20] the most used ones.

For the test executions, a beowulf cluster was built which consisted of three nodes: one master node for distributing jobs and monitoring cluster activity (using *ClOver* main engine), the other two nodes were exclusively installed and used for computation:

Master node: Intel Board SCB2S with two Pentium III processors (1.2GHz, 512KB cache), 256M of RAM and 9GB SCSI hard disk.

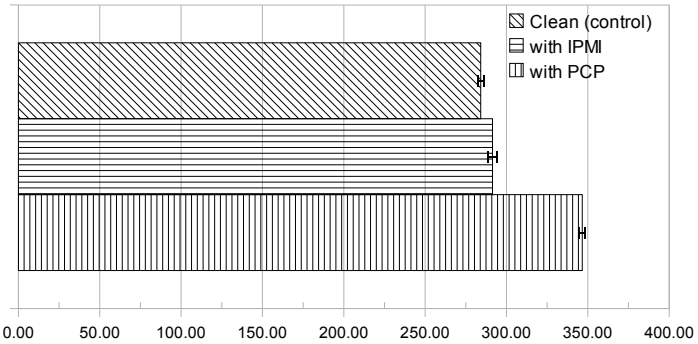


Fig. 5. Wall-time of heat distribution in a pipe solution for three cases: control, with PCP and with IPMI. Hyperthreading was disabled to avoid distortions in the time measurements.

Compute node: Intel Server Board SE7501WV2S with two Xeon (2.66GHz, 512KB cache), 1GB of RAM and 40GB IDE hard disk.

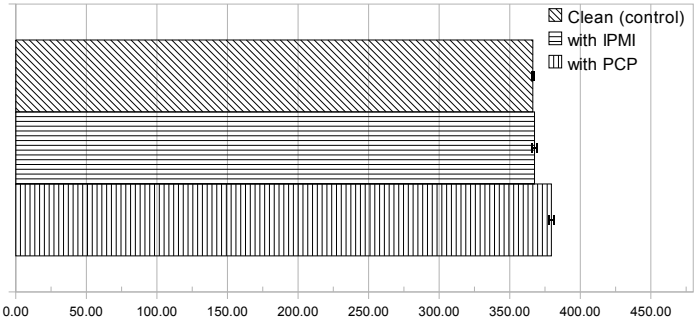


Fig. 6. Wall-time elapsed for solution of heat equation using MPI: control, with PCP and with IPMI.

Again, the selected test application is the solution of the one dimensional heat equation using the finite difference method [21] using OpenMPI. Figure 6 presents the mean wall-time showing no significant difference between control and IPMI runs. On the other hand, PCP runs show a significant increment in the mean wall-time elapsed computing the solution, suggesting an important increase of CPU consumption.

5 Conclusions

Big-scale cluster monitoring has become a non-trivial task as a consequence of the evolution of computational power and storage capacity during the last decade

with the proliferation of *Cluster Computing*. As a result, specialized protocols capable of providing this information in real-time without degradation of performance and avoiding the consumption of computational power have emerged.

Several monitoring solutions have been developed with different features, e.g. Supermon, Hawkeye, Nagios, Ganglia and Clumon. *ClOver*, a new resource monitoring system based on CluMon focused on showing the cluster state at a glance, was introduced. The main engine was re-engineered and re-implemented completely; the plugins scheme was designed to allow simple addition of new features following the main ideas of Nagios.

A new plugin based in IPMI protocol was presented showing good performance results compared to the previous embedded mechanism, i.e. PCP. Both, shared and distributed memory architectures presented similar results where the IPMI plugin showed a significant benefit over PCP in CPU consumption.

The results obtained suggested that the use of specialized hardware protocols for sensoring and monitoring would save valuable CPU cycles.

Acknowledgments

We thank anonymous referees for their valuable comments and D. González Márquez for helping with the drawings. E.M. has a scholarship from CONICET. D.F.S has a scholarship from Fundación YPF. D.M. has a scholarship from IBM. This work was partially supported by grants from IBM Corporation, CONICET (PIP 5756/05) and Universidad de Buenos Aires (UBACyT X132/08).

Acronyms

API Application Programming Interface 7
BMC Baseboard Management Controller 5
GUI Graphical User Interface 4
IPMB Intelligent Platform Management Bus 5
IPMI Intelligent Platform Management Interface 1
LAN Local Area Network 5
MPI Message Passing Interface 8
NCSA National Center for Supercomputing Applications 1
PBS Portable Batch System 4
PCP Performance Co-Pilot 3
POSIX Portable Operating System Interface for Unix 5

References

[1] Top500 supercomputer site, last visited on 05/12/2009 (updated every 6 months). URL http://www.top500.org/

[2] Intelligent platform management interface, last visited on 05/12/2009. URL http://www.intel.com/design/servers/ipmi/

[3] M. J. Sottile, R. G. Minnich, Supermon: A high-speed cluster monitoring system, in: CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing, IEEE Computer Society, Los Alamitos, CA, USA, 2002, p. 39. doi:10.1109/CLUSTR.2002.1137727.

[4] D. Thain, T. Tannenbaum, M. Livny, Concurrency and Computation: Practice and Experience 17 (2–4) (2005) 323. doi:10.1002/cpe.938.

[5] Hawkeye. a monitoring and management tool for distributed systems, last visited on 07/09/2009.
URL http://www.cs.wisc.edu/condor/hawkeye/

[6] Nagios web site, last visited on 07/09/2009.
URL www.nagios.org

[7] Ganglia: open source monitoring system, last visited on 07/09/2009.
URL http://ganglia.sourceforge.net

[8] M. Massie, B. Chun, D. Culler, Parallel Computing 30 (7) (2004) 817. doi:10.1016/j.parco.2004.04.001, [link].
URL http://www.sciencedirect.com/science/article/B6V12-4CMHWWX-2/1/4f25bfa7f38fd02a3fb97adfd3109b85

[9] Clumon, the cluster monitoring system., last visited on 07/09/2009.
URL http://clumon.ncsa.uiuc.edu/

[10] Silicon graphics performance co-pilot, last visited on 05/12/2009.
URL http://oss.sgi.com/projects/pcp/

[11] M. J. Quinn, Parallel Programming in C with MPI and OpenMP, McGraw-Hill Education (ISE Editions), 2003.

[12] B. Chapman, G. Jost, R. van der Pas, Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation), The MIT Press, 2007.

[13] The OpenMP API specification for parallel programming, last visited on 05/12/2009.
URL http://openmp.org

[14] Beowulf organization, last visited on 05/12/2009.
URL http://www.beowulf.org

[15] J. Dongarra, T. Sterling, H. Simon, E. Strohmaier, Computing in Science & Engineering 7 (2) (2005) 51. doi:10.1109/MCSE.2005.34.

[16] P. S. Pacheco, Parallel Programming with MPI, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[17] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, T. S. Woodall, Open MPI: Goals, concept, and design of a next generation MPI implementation, in: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, 2004, pp. 97–104.

[18] Argonne national laboratory, mpich2, last visited on 05/12/2009.
URL http://www.mcs.anl.gov/mpi/mpich2

[19] W. Gropp, E. Lusk, , A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, 2nd Edition, MIT Press, 1999.

[20] W. Gropp, E. Lusk, , R. Thakur, Using MPI-2: Advanced Features of the Message-Passing Interface, 2nd Edition, MIT Press, 1999.

[21] G. Marshall, Solución Numérica de Ecuaciones Diferenciales. Tomo II: Ecuaciones en Derivadas Parciales, Editorial Reverté S.A., Buenos Aires, 1986.