



Efficient use of parallel libraries on heterogeneous Networks of Workstations [☆]

A. Clematis ^{a,*}, G. Dodero ^b, V. Gianuzzi ^b

^a *Inst. per la Matematica Applicata, Consiglio Nazionale delle Ricerche, Via De Marini 6 (Torre di Francia), 16149 Genova, Italy*

^b *DISI – Università, Via Dodecaneso 35, 16146 Genova, Italy*

Abstract

The paper is motivated by efficiency considerations about porting mathematical software from Massively Parallel Processors (MPPs) to Networks of Workstations (NOWs). Heterogeneity of the network is the major obstacle to efficient porting: it can be overcome with a specialized system, Programming Environment for Network of Computers (PINCO), for monitoring available computational power at different nodes, both statically and dynamically. The structure and functionalities of PINCO are outlined, and a significant porting example, matrix multiplication, is presented. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Network of Workstations; Parallel libraries; Heterogeneous networks; Program portability; Load balancing

1. Introduction

Various research fields rely on efficient solutions of computationally challenging problems, which require some form of parallel execution to achieve acceptable efficiency. Quite often, sequential solutions make use of mathematical libraries designed for efficiently solving computational problems in linear algebra, finding the eigenvalues of a matrix, or solving partial differential equations. Use of these routines provides portability and ease of maintenance. There has been much interest recently in developing parallel versions of some se-

quential subroutines package for distributed memory concurrent computers, usually optimized for a given Massively Parallel Processor (MPP). The use of parallel libraries allows to reduce execution timings of existing sequential programs, without even modifying them, that is without costly revisions of their structure for a parallel implementation.

Some of these packages, like for example Parallel Block Basic Linear Algebra Subprogram (PBLAS) [4], have been implemented with such efficiency that the performance of a parallel program implemented by a novice using it, will generally be comparable with that hand-coded by an experienced programmer. Most parallel libraries use the Single Process Multiple Data (SPMD) programming model, where several processes execute the same code, each one on a given data subset. Data distribution over the processors of a concurrent computer has a major impact on the load balance and communication characteristics of

[☆] This work has been partially supported by a grant of Italian CNR under coordinated project “Virtual Libraries for Computational Problem Solution”.

* Corresponding author.

E-mail addresses: clematis@ima.ge.cnr.it (A. Clematis), dodero@disi.unige.it (G. Dodero), gianuzzi@disi.unige.it (V. Gianuzzi).

the concurrent algorithm, and hence largely determines its performance and scalability. For this reason great effort is devoted to data partitioning, in order to give each process the same amount of computation.

Usually, these libraries are implemented for MPP platforms, making use of the respective custom message passing libraries. It is however possible to find implementations making use of a general purpose communication library, such as PVM and MPI, which may run across most UNIX platforms. The availability of these high level communication libraries and their binding with C, C++, and Fortran provide the basic tools to solve the problem of program portability across different parallel and distributed architectures. Running some code developed for architecture A on architecture B is often just a matter of recompiling it. However, the efficiency of a code originally developed for an MPP is seldom maintained, when ported to a workstation network and vice versa, since performance mostly depends on the underlying architectural features.

Technology evolution has made it possible to exploit Networks of Workstations (NOWs) as efficient supercomputing tools. High speed networks connecting increasingly faster systems may now substitute expensive parallel supercomputers at a fraction of their cost. In our perspective an MPP architecture has a regular topology with fast communication links, a large number of homogeneous processors (in the order of some hundreds and more), and it is a dedicated environment which provides to the application the exclusive use of a partition of the whole system, up to its completion. On the other hand, a NOW adopts Ethernet (or similar) communication network (which may be switch or bus based and provides a peak bandwidth of 10/100 Mbit), and consists of some tens of heterogeneous computing nodes which are shared among different applications, in multitasking.

The possibility of reusing parallel libraries developed for MPPs in the NOW environment represents an attractive possibility which in principle is already attainable, but in practice it provides poor performance, since heterogeneity may well result in unbalanced computations. To achieve

efficient implementations, differences in computing power must be taken into account, both as static raw power, and as dynamic workload resulting from jobs submitted by other users. As a result, parallel software for NOWs must be developed either using a non-SPMD model, or SPMD with irregular data partitioning, or SPMD with allocation of as many parallel jobs per processor as suggested by the respective computing power. A discussion and a comparison about the relative performance and applicability of these solutions is presented in Section 2.

A distributed programming environment with a specialized competence in scaling and locating parallel applications, able to maintain load balancing among heterogeneous workstations, is needed, to obtain the maximum efficiency yet preserving the simplicity of using the existing SPMD code. There are few environments that support an easy-to-use access to heterogeneous distributed systems, where mathematical libraries are available, with transparent access to resources. Usually, either they have a complex user interface, or they have limitations in architectural support. The goal of providing and supporting a Parallel Virtual Library for arbitrary NOWs, including the heterogeneous ones, is still far from being achieved. This research area is thus a most promising one where novel systems are now being proposed. A review of existing systems which provide useful features for users of parallel libraries is presented in Section 3.

The system we are developing has been named Programming envIRONMENT for Network of COmputers (PINCO); it is described in [7]. PINCO provides:

- Support for porting library functions to heterogeneous NOWs: adapting them to dynamic workload changes requires only minimal source code modifications.
- Run time support for load monitoring at each moment, to guarantee the best execution performance. The run-time environment is a C distributed program implemented on top of PVM [11], which continuously monitors the net, evaluating the actually available computing power.
- A simple but effective compiling system which is able to automatically generate code for any set

of target architectures, running various UNIX-like OS. The only assumption is that a common file system can be accessed by all nodes in the network.

- A set of programs running some well-known benchmarks is also provided, in order to evaluate the total raw computing power of the NOW.
- An application programmer's interface which allows to a process to start a parallel execution to request for a given computing power, expressed in some unit of measure (we will use the concept of Pentium-equivalence), possibly distributed over a given number of workstations. The system returns a list of available nodes with the respective computing power. Allocation of processes to computing nodes is selected by the starting process and executed by the system.

This situation allows implementers to tune their parallel code by experiencing with different allocation strategies. Even if PINCO is rather rudimentary in that respect, since no automatic allocation is provided yet, following versions will benefit from these investigations and implement more sophisticated and performant strategies.

Section 4 describes PINCO architecture, while Section 5 illustrates the programming environment for heterogeneous networks. Finally, figures of merit about PINCO are presented in Section 6, considering a matrix multiplication algorithm.

2. Porting efficient SPMD programs to heterogeneous NOWs

A key aspect to exploit the computing resources of a given architecture is to adopt a suitable programming model. The SPMD model is widely used on MPPs, with a regular data decomposition structure which reflects the logical structure of the problem, and may be directly supported by the physical MPP communication network. Process interactions are normally loosely synchronous ones, with non-blocking send and blocking receive, and the processes are characterized by a cyclic execution of computation and communication phases, with implicit or explicit loose-synchronization at the end of each cycle.

A simple recompilation of SPMD regular processes on heterogeneous NOWs with workload varying across time yields poor performance, since the slowest process/processor synchronizes the whole execution. Thus, it is necessary to adapt the workload of regular SPMD algorithms to a distributed non-dedicated platform. This can be achieved as follows:

1. SPMD model with regular partition. Load balancing is achieved by allocating more than one process per processor.
2. SPMD model with irregular data distribution, allocating one process per processor. Since the workload at each node is a function of the amount of data contained in each subdomain, we can keep the workload balanced by means of an irregular data distribution, inversely proportional to other load factors.
3. Master/Worker model, with a bag of symmetric tasks and one process per processor. The cardinality of processes is usually lower than that of tasks.

As for coding effort, the first model usually requires just a few minor modification to source code of existing SPMD programs, yet it adds some overhead for context switching among parallel processes on the same processor. Such overhead can, to some extent, be reduced using threads rather than processes, and grouping together send and receive operations to decrease context switching timings. The second model may require some revision to the algorithm, to allow variable data partitioning inversely proportional to actual workload on the node where each process is allocated. On the other hand, the Master/Worker model is often asynchronous and self-balancing, especially when the number of processes is significantly smaller than that of tasks. However data communication timings may considerably increase, because of data packet latency at each exchange.

Both the first two models require an efficient support for balancing SPMD computations, capable of maintaining efficiency even when the platform is subject to highly dynamic variation. In other words, process or data migration from overloaded to underloaded nodes may be needed, and programs must be written "parametrically",

which is easier to do with the first programming model rather than with the second. The advantage of the second model, with respect to the first one, is to eliminate context switch overhead for processes on the same node. If fine grain data partitioning is possible for the given problem, this model allows a better match to available computing power.

As remarked in [9], “programs written using SPMD model are often moldable. With moldable jobs, the number of processors is set at the beginning of execution, and the job initially configures itself to adapt to this number. After it begins execution, the job cannot be reconfigured. The same application may of course run several times using different partition sizes” selecting the most favorable one in accordance with present target load. Moreover, schedulers that oblige to use different programming styles “may thus alienate users that would rather use another style. This applies, for example, to schedulers that require all jobs to be able to adapt to changes in resource allocation at run time, something that is difficult to achieve in certain cases.” For all these reasons we selected the first model as the preferred one. Results of comparisons on the matrix multiplication problem, implemented with the above three models, are presented in Section 6 as support for such a choice.

3. Related works

The idea of using distributed resources and remotely executing library functions is implemented by NetSolve [1], a net server for the solution of computational problems developed by the Jack Dongarra’s group. NetSolve allows users to access computational resources, such as hardware (workstations) and software (math function libraries), distributed across the network. It has been designed in order to allow users (including remote users) to access libraries not available on their networks for lack of code or license. Access may take place by means of function calls inside C++ or Fortran programs, or interactively via a Java-based graphical interface.

The Netsolve system is a set of loosely connected machines, that is, the machines can be on

the same local network or on an international network. Clients issue a call to the Netsolve system by specifying the “problem” to solve (e.g. matrix multiplication) and required parameter values. An agent is contacted, which yields back a list of available computational servers, in decreasing order of efficiency. Then, the call is suitably coded and sent to the first server in the list. Support for parallel libraries development is not considered inside Netsolve. On the other hand, PINCO supports development of parallel functions to be executed on a cluster of workstations with fast communication links, allowing to easily adapt to execution time workload changes, by changing the number of parallel processes to be spawned, thus explicitly supporting SPMD computations, with load balancing for better network resources exploitation.

Well-known load balancing systems for batch jobs are CODINE, by GENIAS, and LSF – Load Sharing Facilities (for more details about the LSF scheduler see [10]), by Platform Computer. They support facilities for batch queuing, job management, deadline scheduling, in a word, the maximum utilization of the resources, at day time and overnight. However, their aim is to reach the net maximum efficiency, rather than offering a specific support for parallel programming, which is possible, but not privileged. It is worth mentioning UTOPIA [12] as well, a load sharing facility implemented on top of Unix for very large and heterogeneous systems. As well as LSF it is a general purpose user transparent system, which supports remote execution only at job initiation time.

Let us consider briefly how LSF considers parallel execution. As part of submitting a job, an LSF user can specify the number of processors required. When LSF finds a sufficient number of processors satisfying the resource constraints for the job, it spawns an application “master” process on one of the processors, passing to this process a list of processors. The master process can then use this list of processors to spawn a number of “slave” processes to perform the parallel computation. The slave processes are completely under the control of the master process, and as such, are not known to the LSF batch scheduling system. PINCO, like Netsolve and LSF, answers the

requesting process with a list of available processors, together with an indication of respective computing power, in order to optimize parallel process allocation. Another difference with respect to LSF is that PINCO operates “interactively” rather than on batch jobs, that is, it is invoked only when an application process calls a library function which uses PINCO for exploiting its parallelism.

Finally, DATA Migration Environment (DAME) [3], is, from certain points of view, the system closest to PINCO, since it aims to dynamically balance the workload of SPMD regular computations with the irregular data partitioning model. The most significant difference is that the reconfiguration protocol is based on data migration instead of process migration, thus requiring the cooperation of the application programmer, which must implement decomposition-independent programs.

4. PINCO architecture

We have already defined the NOWs as basically heterogeneous systems. As discussed in [12], three kinds of heterogeneity can be considered, that is:

- Configuration: hosts have the same architecture but different configuration,
- Architectural: executable files cannot be exchanged, and
- Operating System heterogeneity.

PINCO is considering the first two kinds of heterogeneity, and it allows to use different UNIX-like operating systems. It entirely works at user level, without requiring exclusive use of the network; it is “light” with respect to execution time overhead, installation difficulties, and ease of use.

PINCO has been designed as composed by three parts: the Job Scheduler and the Resource and Task Manager, which provide the related facilities, and the Application Interface which acts as intermediate level between the application and the other two components. Moreover, a programming environment is provided, to help PINCO programmers in developing distributed applications. Keeping system components as much separate as possible is useful to experience on different job and resource management techniques, by substituting

individual components with others implementing alternative policies.

Each parallel job is executed on a partition, that is, on a subset of nodes. Due to the heterogeneity of the hardware architecture, some form of normalization is needed in order to dynamically evaluate and compare the computational power of each node. The unit we choose is the tenth of a Pentium 133 (ETP). That is, at each moment, PINCO assumes to have a virtual pool of Pentium 133 unloaded, and satisfies each request selecting a set of physical machines which offers the greatest number of ETPs (to minimize inter-node communications).

So far we have not taken into account the performance of the underlying network, either as absolute or relative performance (with respect to some conventional unit of traffic). The reason is due to the relatively low impact that such figures may have on our intended applications, which are computation intensive and less sensitive to traffic loads.

Presently, only a part of PINCO has been implemented, that is:

PPE, the PINCO Programming Environment which allows to compile and to allocate both PINCO and the parallel application, and which executes the off-line evaluation of the computing power for each kind of network of workstations (see Sections 5.1 and 5.2);

PRTM, the distributed PINCO Resource and Task Manager, which periodically evaluates the workload of each network component, processes programmer requests for the computing power needed by the applications, and eventually activates the parallel tasks (see Section 5.3).

In case of significant workload changes on some node of the network, as in practice is often the case, provisions for tasks migration must be taken. Reconfiguration may exploit tools supporting task synchronization and process migration, which are already available [5,2].

In order to start experiencing with distributed resource allocation, a simple API has been implemented, by means of a limited set of calls to spawn, control and monitor tasks under the control of PRTM. An application process initiating a parallel execution invokes a function,

PINCO_grant(), specifying the amount of requested computing power and possibly the number of requested workstations. It gets back as an answer a list of available workstations, with the respective computing power. Allocation of parallel processes to workstations is selected by the initiating process by invoking another function, PINCO_spawn. This demands from the programmer the burden of deciding about task allocation: in such a way it is possible to easily experience with different scheduling policies, before embedding them in the system. In other words, the allocation policy is variable [9], since the partition is determined at job submission time, only considering user requests. Thus, PINCO may statically allocate parallel programs, provide the suitable computing power and keep a balanced load on the net.

To achieve maximum system portability, an already available platform must be selected: the choice falls on PVM, a communication library for distributed memory machines, which is becoming the most widely used on NOWs. PVM is also upwards compatible with MPI, the standard for parallel applications, which as present has been less used on heterogeneous networks. The present version of PINCO is implemented on top of PVM, and its architectural model is similar to that of PVM, that is, it is based on the PINCO Master Daemon, running on the master host, usually, the most reliable of the net.

The Master Daemon is a process, unique in the network, composed by three software modules: the

Load Monitor (LM), the Resource Manager (RM) and the Communication Library INterface (CLIN). CLIN includes all the PVM-based communication functions: porting of PINCO to other parallel or distributed machines, with different communication features, is made easy by such encapsulation. LM collects the load situation at each host, while RM is responsible for balancing the workload among multiple execution hosts. RM also acts as Submission Manager: the application sends computing power and task activation requests, obtains the service and a report with other useful information.

On each host, the PINCO Local Daemon is executed. It is composed by three modules: the Local LM, the Process Handler and the Local CLIN. The first one evaluates the local load, which is periodically sent to the Master Daemon. The Process Handler is responsible for activating local tasks, according to the Master Daemon requests, and for sending back information about status of the computation, when needed.

Fig. 1 shows master–local daemon relationships, while Fig. 2 describes Master Daemon inner structure.

The choice of building PINCO on top of an already existing communication library also allows to easily obtain additional and useful features such as signal handling and terminal I/O for each remotely executed process. Remark that application processes may communicate among one another using any library, i.e. Unix sockets, PVM, Linda

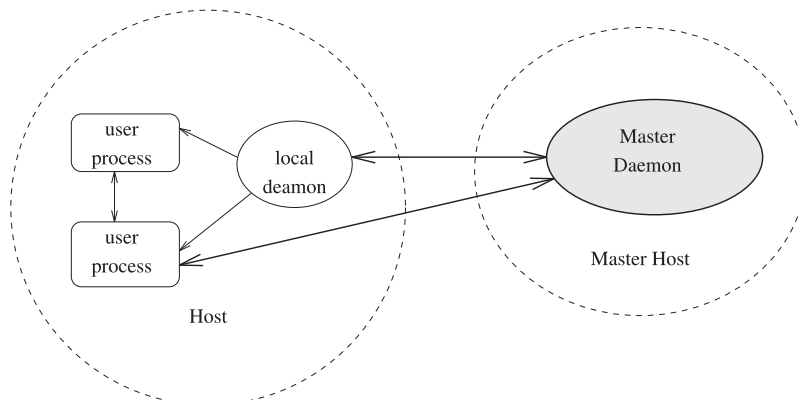


Fig. 1. Daemons relationships.

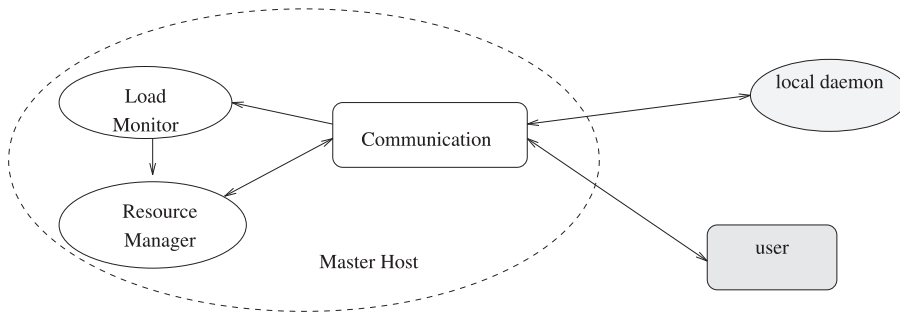


Fig. 2. PINCO Master Daemon structure.

or others, since PINCO does not interfere with application communications.

5. PINCO parallel environment

Considering that PINCO run time environment is made by heterogeneous nodes and the code distribution is decided in a dynamic way, it is necessary to generate code for each possible different target architecture. The word architecture is used in this case as a synonym of operating system version and it identifies executable code compatibility.

Other parallel libraries like PVM, MPI, Linda, take into consideration the possibility of generating code for different target architectures, and this is one of the key points in providing PINCO portability. Most of the work is however left to the user who is charged of the task of recompiling code for each different target architecture he/she wants to include in the parallel machine, and then of placing each copy of executable code in the proper directory. This may become a tedious task at least when the number of heterogeneous architectures is greater than two.

5.1. Compilation environment

The PINCO compilation system generates codes for the different target architectures and distributes it so that it will be accessible from any machine of the parallel system. The user has only to set two parameters:

- a variable which indicates the target application, this is represented by the `PINCO_ROOT` environment variable in Unix systems, and normally indicates the root directory containing the source code of the application;
- a configuration file which contains the names of selected target architectures.

The final file system configuration for the PINCO environment is represented as follows:

```

$PINCO_ROOT
/(PINCO utilities and configuration
  files)
SRC (contains the source code)
INCLUDE (header files)
OBJ (object files sorted by archi-
  tecture)
BIN (binary images sorted by archi-
  tecture)
DAT (application data)
LIB (libraries sorted by architec-
  ture)
  
```

This logical organization may be mapped in different ways on the physical file system, depending on network organization.

5.2. Evaluating system resources

For an efficient management of computing resources available on the local area network, PINCO must be able to evaluate resources, and this is a challenging task since we are considering a heterogeneous and shared environment. Consider first how to evaluate total computing power of the parallel system as the sum of heterogeneous resources. This point could be very subtle: different

machines may be more or less suited for different applications (e.g. matrix multiplication and sorting algorithms), and many factors may determine the actual efficiency of a single node in processing a selected application (CPU speed, memory hierarchies, specialized coprocessors and the like).

PINCO approach to this problem has been experimentally derived by collecting data about a sample target network and a set of suitable benchmarks: Table 1 summarizes the set of available machines while benchmark results are shown in [6]. Running the benchmarks on the different nodes, it is possible to get performance figures for the whole network. These figures are not strictly defined because the different nodes may change their relative performance selecting a different benchmark. In any case it is possible to get an approximate value of relative speeds of the different machines, which may be used in most cases as suitable indication.

PINCO is designed to run on shared systems, hence we have to consider that actual computing power at each node will be only a fraction of the statically computed power of the same node. This fact can be expressed by the following simple formula: $ds_i = (1 - \sigma_i)s_i$, which correlates the dynamic relative speed with the static relative speed of node i . Here the parameter σ_i expresses the load factor of node i . In Unix based systems, a simple

but effective way of evaluating load factors is to use the percentage of CPU usage as provided by ps system call. Actually this is the choice currently implemented inside PINCO.

5.3. PINCO Resource and Task Manager

PINCO architecture described in Section 4 has evidenced that there does not exist a single component which may be called “the Allocator”. Such a functionality is distributed across the network, yet maintaining a centralized information collector, that is the Resource and Task Manager (PRTM). The latter is a process running on the host node, where information sent by the various instances of the Load Monitor is kept, and used to take decisions about process allocation to the network. At present allocation decisions result from negotiation with the user process, since the choice suggested by PRTM is evaluated and partially modified, if needed, by the user. As far as experience in this decision making process is collected, it is expected to include “smart” allocation policies in PRTM, thus making user negotiation no longer needed for most applications.

Information to be collected by the LM can be classified as:

- static info: initialized as collected from off-line benchmarks (see Section 5.2);
- dynamic info: computed and periodically updated by the various local daemons.

Dynamic information is computed by each Local LM, running on a different PINCO host computer. The percentage of CPU free is computed every 5 s by issuing a local system call, such as the command `ps` from a shell. This value is sent again only if different for more than 10% from the one previously sent to the LM. When the Load Monitor receives a message from the Local LM at node x , it updates the so-called `load_table` where dynamic and static network load parameters are kept. The static “weight” of node x is multiplied by the percentage of CPU free, thus giving the available power on node x for PINCO requests. This value is still expressed in ETP.

Table 1
IMA-CNR heterogeneous local area network

Name	Model	OP SYS
Paperoga	Sparc Station 20	OS4.1.3_U1
Athena	Sparc Station 20	Solaris 2.6
Helios	Sparc Station 1	OS4.1.3
Aphrodite	Sparc Station 10	OS4.1.3
Elba	Sparc Classic 4/15	Solaris 2.5
Fred	HP 9000/715	HP-UX 9.01
Venus	Sparc St. Classic	Solaris 2.5
Paperinik	Sparc St. Classic 10	OS4.1.3_U1
Bergeggi	Sparc Station 5	Solaris 2.5
Asterix	UltraSparc 5	Solaris 2.6
Obelix	UltraSparc 5	Solaris 2.6
Gauguin	Indigo 2	IRIX 6.2
Matisse	SGI O2	IRIX 6.3
Kandiski	SGI O2	IRIX 6.3
Baobab	Pentium 133 MHz	Linux

5.4. PINCO application program interface

Allocation requests from user processes are based on a data structure called `grant_record`. It is filled by PRTM and passed to the user process by means of a two phase protocol:

1. in the first phase, the user process issues a request expressed by the call

```
err = PINCO_grant(requestedpower,
                 grant_record);
```

2. the user process may check compliance of the proposed allocation to possible additional constraints and then issues the activation request:

```
err = PINCO_spawn(executablefile,
                 grant_record);
```

The function `PINCO_grant` fills the following data structure, in accordance with values actually stored in the `load_table`:

```
struct grant_record
{ float grant_power; /* granted power
 */
  int cluster_dim; /* host number */
  node * topology;
} grant_record;
```

where `node` is a list of `cluster_dim` pairs (`processor_name`, `local_granted_power`), such that the sum of all `local_granted_power` fields, as stored in `grant_power`, matches the request or default approximates it if it is impossible.

Loads corresponding to `local_granted_power` are reserved for the process that invoked the `PINCO_grant`, but parallel processes are not immediately spawned. This happens only when a `PINCO_spawn` is invoked by the user process, where the actual number of parallel processes per processor is specified, by adding a third value in the node list of the `grant_record`. In case the user process does not issue a `PINCO_spawn` within a predefined delay, reserved resources are however freed. The user process may also update the `grant_record`, before invoking a `PINCO_spawn`, in order to use just a subset of available resources, e.g., by not using certain nodes, or

leaving some fraction of computing power to other applications.

6. An example: Efficient porting of a matrix multiplication algorithm

Matrix multiplication ($C = A \times B$) is part of the computational kernel of different linear algebra libraries, its high cost makes it suitable for parallelization and different parallel versions have been described in literature, here we focus our attention on two different algorithms:

- *SPMD* algorithm using a ring of processes. Matrix A is equally partitioned into groups of contiguous rows, each group is distributed to a process. Groups of rows of matrix B are initially equally partitioned among processes and then are circulated along the ring so that each group visits all processes. When a process receives a group of rows of B it sends it immediately to the next process and then computes a part of its local product.
- *Master/Worker* or bag-of-tasks algorithm. Matrix A is partitioned into equally sized groups of rows, and matrix B is partitioned into equally sized groups of columns. A group of rows and columns is a task. The master distributes tasks to workers. Workers asynchronously execute local multiplication and grasp new tasks until the bag is empty.

The SPMD algorithm is well-suited for MPP and in this or in some close version (like Pipe-Multiply and Roll) is often included in parallel numerical libraries. The Master-Worker ensures good performance on heterogeneous NOW, being self-balancing, provided that a suitable task granularity is adopted. Optimal task granularity is obtained considering a trade-off between communications, whose cost increases when fine grain partitioning is adopted, and load-balancing, which depends on the heterogeneity of the computing nodes and on the ratio “process number/data partition cardinality”, and which normally worsens when coarse grain is selected. To get an efficient version of this type of algorithm, a supplementary programming effort is needed, aimed at making parametric code with respect to

Table 2

Data about the subnetwork used for matrix multiplication IMA-CNR heterogeneous local area network

Workstation	Relative speed	Incremental computing power of subnetwork	Execution time (s) for the sequential MM on 1000×1000 matrices
Athena	1.00	1.00	1119.60
Matisse	1.87	2.87	598.71
Kandinsky	1.90	4.77	589.26
Obelix	1.87	6.64	598.69
Asterix	1.85	8.49	605.18
Gauguin	5.87	14.36	190.73

the number of computing nodes (that is the number of workers) and the granularity of the partition. We are not aware of numerical libraries which include this version of matrix multiplication.

We have used a sub-network of IMA NOW with up to six computing nodes to compare performances of the two algorithms. Table 2 reports the name, the relative speed of the computing elements, the incremental computing power of the configuration (from two to six nodes) and the execution time for the sequential algorithm. All timings in this section refer to multiplication of two square matrices of 1000 rows and columns of double. Note that a relative speed of 1 is assigned to the slowest node. In Fig. 3, the measured speed-up for the two versions of the code is reported.

The Master–Worker version of the code provides very good performance, reaching speed-up values which are more than 90% of the ideal speed-up, but it has to be written specifically for NOW. The SPMD version permits a complete reuse of code available for MPP systems but it shows poor performance, since it is based on regular and uniform data distribution thus resulting in constraint by the slowest node.

Consistently with the considerations of Section 2 we have tried two possible strategies to improve the SPMD algorithm:

- First of all the Ring-oriented procedure has been implemented with Non-Regular Data Distribution (NRDD) under user control (this version required an extra effort in programming which can be quantified in an increase of lines of code of about 30%).
- Finally the Ring-oriented procedure has been implemented with a Variable Number of Pro-

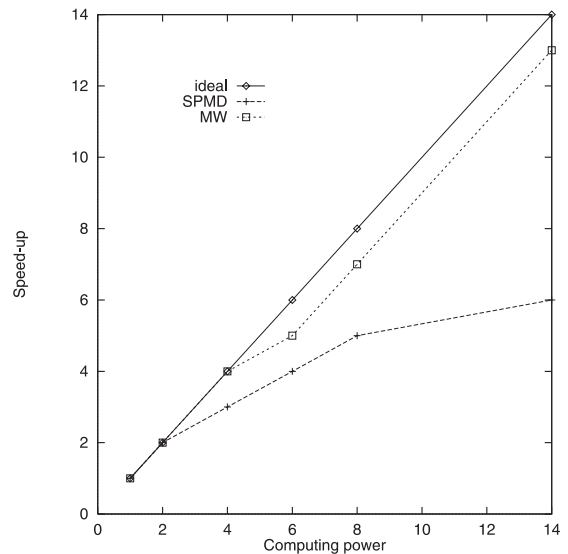


Fig. 3. Speed-up vs. computing power for MW and SPMD.

cesses (VNP) on each node, and regular data partitioning. In this second operation we rely on PINCO in order to execute on each node a number of processes proportional to the relative speed of the node. In this case no intervention has been necessary and we completely reused the SPMD code.

Fig. 4 reports speed-up values for these two versions: the problem due to the unbalancing effect of regular data distribution on heterogeneous nodes disappeared in both version and the obtained speed-up is close to the ideal one.

The following considerations can be derived from this experience:

- MW code looks attractive since it is self balancing and does not require any run-time perfor-

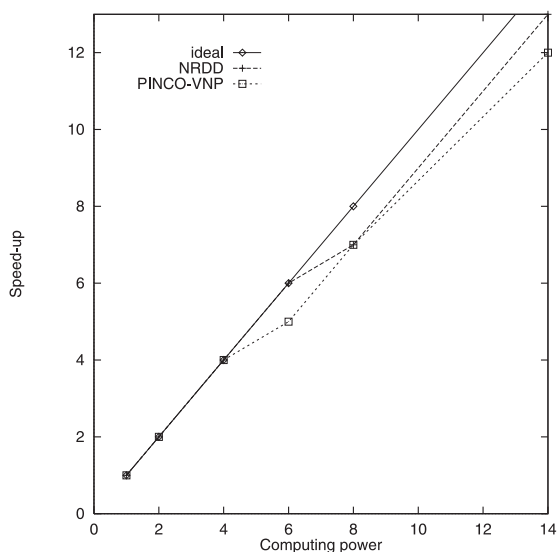


Fig. 4. Speed-up vs. computing power for NRDD and VNP.

mance support. However, to develop a master worker version of a given parallel algorithm code may require a coding effort which may be afforded if the code has to be written from scratch, but it may be considered too high if other parallel implementations of the same algorithm are available. Also, for some algorithm it is very difficult (or even impossible) to develop a pure Master–Worker version. In this situation it may be attractive to use SPMD versions without losing performance.

- Both data irregular distribution and variable process number SPMD models need performance monitoring run-time support to decide the optimal distribution. Using PINCO, and automatically generating and allocating the most suitable number of tasks, it is possible to reach the same speed-up of the other two models, yet fully reusing the original SPMD implementation, without intervention on the code.

Fig. 5 summarizes this situation providing relative efficiency for the four different versions of the algorithm.

This example shows that PINCO is a useful tool for heterogeneous network management which has the nice feature of improving efficient portability of SPMD programs to heterogeneous environment.

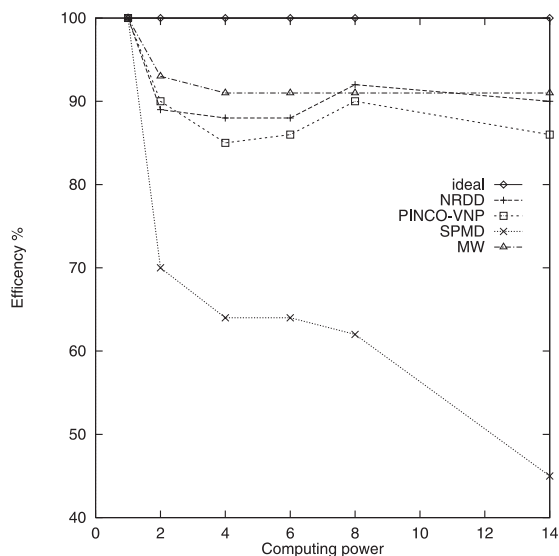


Fig. 5. Efficiency for different implementations.

7. Conclusions and future works

In this paper, we have described PINCO, a system for developing efficient implementations of parallel library routines on heterogeneous NOWs. Emphasis has been given as well to methods for porting existing libraries, developed for MPPs, and usually employing SPMD programming models, without much coding effort and without loss of efficiency. At present, PINCO runs on two networks, the one at IMA-CNR and the student lab at DISI. The first one is highly heterogeneous, both in configuration and architecture, with different Unix-like operating systems. The second one is composed by about 45 Pentiums, from 133 to 350 MHz, with different configurations, all of them running Linux. Efficiency measures presented in the paper have been taken on the most heterogeneous network at IMA. A significant example has been presented, that is how to preserve efficiency in distributing a parallel implementation of matrix multiplication to a heterogeneous NOW. The key point of PINCO modeling of the heterogeneous resources lies in the ability to evaluate both statically and dynamically the “available computing power” of the NOW, which is taken as input for balancing computational load caused by parallel library functions.

A more significant effort, still under way, is porting to heterogeneous NOWs of the PB-BLAS package, as an intermediate step of porting the widely used Basic Linear Algebra Subprograms (BLAS) library. PB-BLAS allow to perform dense linear algebra computations on block-partitioned matrices. Its use is restricted to computations where one or more of the matrices involved consist of a single row or column of blocks.

Vectors and matrices to be used as arguments are evenly distributed among available processors. The PB-BLAS consist of calls to the sequential BLAS for local computations, and calls to the Basic Linear Algebra Communication Subroutines (BLACS) [8] for communication. BLACS provide communication routines for sending and receiving parts of matrices between two or more processors, global reduction routines in which all processors take part, and few general support routines for setting up the communication network. BLACS have been also implemented on top of PVM. In this implementation a function, named BLACS_SETUP, allows to actually allocate the Parallel Virtual Machine and spawn off the processes, in a Round Robin fashion. Knowing actual computational power of each node enrolled in PVM at run time, better load balancing is achieved, by performing suitable allocation. By using versions of PVM which already allow process migration, like [2], load may be rebalanced even during parallel execution. Such a feature would impact on process spawning and identification inside BLACS, without affecting the computational portion of the library (BLAS and PB-BLAS) which would remain untouched.

References

- [1] H. Casanova, J. Dongarra, Netsolve a network-enabled server for solving computational science problems, *The International Journal of Supercomputer Applications and High Performance Computing* 11 (3) (1997) pp. 212–223.
- [2] J. Casas, R. Konoru, S.W. Otto, R. Prouty, J. Walpole, Adaptive load migration systems for PVM, in: *Proceedings of the Supercomputing'94*, 1994, pp. 390–399.
- [3] M. Cermele, M. Colajanni, G. Necci, Dynamic load balancing of distributed SPMD computations with ex-

PLICIT message-passing, in: *Proceedings of the Sixth Heterogeneous Computing Workshop*, IEEE Computer Society Press, Los Alamitos, 1997, pp. 2–16.

- [4] J. Choi, J.J. Dongarra, D.W. Walker, PB-BLAS: a set of parallel block basic linear algebra subprograms, Technical Report ORNK/TM-12468, 1994.
- [5] A. Clematis, G. Deconinck, V. Gianuzzi, A flexible state-saving library for message-passing systems, in: *Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing*, IEEE Computer Society Press, Los Alamitos, 1998, pp. 335–341.
- [6] A. Clematis G. Doderò, V. Gianuzzi, PINCO: system design and examples of use, Technical Report IMA-CNR-8-1998.
- [7] A. Clematis G. Doderò, V. Gianuzzi, A resource management tool for heterogeneous networks, in: *Proceedings of the Seventh Euromicro Workshop on Parallel and Distributed Processing*, IEEE Computer Society Press, Los Alamitos, 1999, pp. 367–373.
- [8] J.J. Dongarra, R.A. van de Geijn, Two dimensional basic linear algebra communication subprograms, LAPACK Working Note 37, Technical Report CS-91-138, University Of Tennessee, 1991.
- [9] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, P. Wong, Theory and practice in parallel job scheduling, in: *IPPS'97 Workshop Job Scheduling Strategies for Parallel Processing*, 1997, pp. 1–34.
- [10] E.W. Parsons, K.C. Sevcik, Implementing Multiprocessor Scheduling Disciplines, in: *Third Workshop on Job Scheduling Strategies for Parallel Processing*, 1997, pp. 166–192.
- [11] V.S. Sunderam, G.A. Geist, J. Dongarra, R. Manchek, The pvm concurrent system: evolution experiences and trends, *Parallel Computing* 20 (4) (1994) pp. 531–546.
- [12] S. Zhou, J. Wang, X. Zheng, P. Delisle, UTOPIA: a load sharing facility for large, heterogeneous distributed computer systems, *Software – Practice and Experience*, vol. 23, Wiley, UK, 1993, pp. 1305–1336.



Andrea Clematis received an advanced degree in Mathematics in 1982 from the University of Genova. He is a senior Researcher, and head of parallel computing group at the Istituto per la Matematica Applicata of Consiglio Nazionale delle Ricerche. He is with CNR since 1984. In the period from 1988 to 1994 he was also an Adjunct Professor at the Computer Science Department of the University of Genova Italy, and in summer 1988 a Visiting Associate Professor at Department of Computer Science and Engineering, University of Nebraska Lincoln, USA. In the period from 1983 to 1984 he was a consultant for TXT – Techint Software Telematica Milano – Italy. His research interest include programming environments and languages, high performance computing methodologies and applications.



Gabriella Dodero received an advanced degree in Mathematics from the University of Genova 1977. She was a research associate of CNR, the Italian National Research Council, until 1981. She now is Associate Professor at DISI the Computer Science Department in the University of Genova, Italy. Her research interests mainly focus on parallel and distributed applications, and mobile networking.



Vittoria Gianuzzi received an advanced degree in Mathematics in 1975 from the University of Genova. She was a research associate of CNR, the Italian National Research Council until 1981. She now is a Senior Researcher at DISI, the Computer Science Department in the University of Genova, Italy, and is National Coordinator for the CNR project on parallel Virtual Libraries. Her research interests include distributed and fault tolerant systems.