

# The MOL Project: An Open, Extensible Metacomputer\*

A. Reinefeld, R. Baraglia<sup>†</sup>, T. Decker, J. Gehring,  
D. Laforenza<sup>†</sup>, F. Ramme, T. Römke, J. Simon

PC<sup>2</sup> – Paderborn Center for Parallel Computing, Germany

<sup>†</sup> CNUCE – Institute of the Italian National Research Council

## Abstract

*Distributed high-performance computing—so-called metacomputing—refers to the coordinated use of a pool of geographically distributed high-performance computers. The user's view of an ideal metacomputer is that of a powerful monolithic virtual machine. The implementor's view, on the other hand, is that of a variety of interacting services implemented in a scalable and extensible manner.*

*In this paper, we present MOL, the Metacomputer Online environment. In contrast to other metacomputing environments, MOL is not based on specific programming models or tools. It has rather been designed as an open, extensible software system comprising a variety of software modules, each of them specialized in serving one specific task such as resource scheduling, job control, task communication, task migration, user interface, and much more. All of these modules exist and are working. The main challenge in the design of MOL lies in the specification of suitable, generic interfaces for the effective interaction between the modules.*

## 1 Metacomputing

*“Eventually, users will be unaware they are using any computer but the one on their desk, because it will have the capabilities to reach out across the national network and obtain whatever computational resources are necessary” [41]. This vision, published by Larry Smarr and Charles Catlett in their seminal CACM article on metacomputing, sets high expectations: “The metacomputer is a network of heterogeneous, computational resources linked by software in such a way that they can be used as easily as a personal computer.”*

---

\*This work is partly supported by the EU ESPRIT Long Term Research Project 20244 (ALCOM-IT) and by the Northrhine Westphalian Initiative “Metacomputing: Distributed Supercomputing”

The advantages of metacomputing are obvious: Metacomputers provide true supercomputing power at little extra cost, they allow better utilization of the available high-performance computers, and they can be flexibly upgraded to include the latest technology. It seems, however, that up to now no system has been built that rightfully deserves the name ‘metacomputer’ in the above sense. From the user's point of view, the main obstacles are seen at the system software level, where non-standardized resource access environments and incompatible programming models make it difficult for non-experts to exploit the available heterogeneous systems. Many obstacles in the cooperative use of distributed computing systems can be overcome by providing a homogeneous, user-friendly access to a reliable and secure virtual metacomputing environment that is used in a similar way as a conventional monolithic computer today.

Some of these issues are addressed by “Metacomputer Online (MOL)”, an initiative that has been founded with the goal to design and implement the nucleus of a practical distributed metacomputer. MOL is part of the Northrhine-Westphalian Metacomputer Initiative that has been established in 1995, and it is embedded in several other European initiatives [31].

The MOL group has implemented a first, incomplete ‘condensation point’ of a practical metacomputer, which is now being extended and improved. Clearly, we could not tackle all relevant obstacles at the same time. We initially concentrated on the following issues that are deemed most important in the design of a first prototype:

- provision of a generic, user-friendly resource access interface,
- support of interoperability of existing codes using different message passing standards,

- effective global scheduling of the subsystems for high throughput and reduced waiting times,
- support of concurrent use in interactive and batch mode,
- mechanisms for automatic remote source code compilation and transparent data distribution,
- automatic selection of adequate compute nodes from a pool of resources to be assigned to the tasks of a parallel application,
- dynamic re-placement of user tasks by means of performance prediction of the source code on the heterogeneous nodes,
- provision of data management libraries and programming frames to help non-expert users in program design and optimal system utilization.

Clearly, this list is not complete. Further items can and should be added for a full metacomputing environment. Depending on their attitude and current task, users might have different expectations on the services a metacomputer should provide. As a consequence, a metacomputer cannot be regarded as a closed entity, but it is rather a highly dynamical software system that needs continuous adaptation to the current user demands.

The MOL project aims at integrating existing software modules in an open, extensible environment. Our current implementation supports PVM, MPI and PARIX applications running on LAN- or WAN-connected high-performance computers, such as Parsytec GC, Intel Paragon, IBM SP2, and UNIX workstation clusters.

This paper presents the current status of MOL. It gives an overview about the system architecture and it illustrates how the separate modules interact with each other. Section 2 reviews related work which influenced the design of MOL or which can be integrated into MOL at a later time. Section 3 gives a conceptual view and discusses the general MOL architecture. In Section 4, we elaborate on the interface design and show how the MOL components interact with each other. Section 5 to 7 describe the MOL components in more detail, and Section 8 gives a brief summary and outlook.

## 2 Previous Work

In the past few years, several research projects have been initiated with the goal to design a metacomputer

environment. Some of them follow the top-down approach, starting with a concrete metacomputing concept in mind. While this approach seems compelling, it usually results in a “closed world metacomputer” that provides a fixed set of services for a well-defined user community.

The second category of projects follow the bottom-up approach, initially focusing on some selected aspects which are subsequently extended towards a full metacomputing environment. In the following, we review some projects falling into this category.

*Parallel programming models* have been a popular starting point. Many research projects targeted at extending existing programming models towards a full metacomputing environment. One such example is the Local Area Multicomputer *LAM* developed at the Ohio Supercomputer Center [13]. *LAM* provides a system development and execution environment for heterogeneous networked computers based on the MPI standard. This has the advantage that *LAM* applications are source code portable to other MPI systems. The wide-area metacomputer manager *WAMM* developed at CNUCE, Italy, is a similar approach based on PVM [5, 6]. Here, the PVM programming environment has been extended by mechanisms for parallel task control, remote compilation and a graphical user interface. Later, *WAMM* has been integrated into *MOL* and extended to the support of other programming environments, as shown below.

*Object oriented languages* have also been proposed as a means to alleviate the difficulties of developing architecture independent parallel applications. *Charm* [26], *Trapper* [38], *Legion* and *Mentat* [25] for example, support the development of portable applications by object oriented parallel programming environments. In a similar approach, the *Dome* project [2] at CMU uses a C++ object library to facilitate parallel programming in a heterogeneous multi-user environment. Note, however, that such systems can hardly be used in an industrial setting, where large existing codes (usually written in Fortran) are to be executed on parallel environments.

Projects originating in the *management of workstation clusters* usually emphasize on topics such as resource management, task mapping, checkpointing and migration. Existing workstation cluster management systems like *Condor*, *Codine*, or *LSF* are adapted for managing large, geographically distributed ‘flocks’ of clusters. This approach is taken by the Iowa State University project *Batrun* [42], the Yale University *Piranha* project [14], and the Dutch *Polder* initiative [34], both emphasizing on the utilization of

idle workstations for large-scale computing and high-throughput computing. Complex simulation applications, such as air pollution and laser atom simulations have been run in the *Nimrod* project [1] that supports multiple executions of the same sequential task with different parameter sets.

These projects could benefit by the use of special metacomputing schedulers, such as the Application-Level Scheduler *AppLeS* developed at the University of California in San Diego [9]. Here, each application has its own AppLeS scheduler to determine a performance-efficient schedule and to implement that schedule in coordination with the underlying local resource scheduler.

*Networking* is also an important issue, giving impetus to still another class of research projects. *I-WAY* [22], as an example, is a large-scale wide area computing testbed connecting several U.S. supercomputing sites with more than a hundred users. Aspects of security, usability and network protocol are among the primary research issues. Distributed resource brokerage is currently investigated in the follow-up *I-Soft* project. In a bilateral project, Sandia’s massively parallel Intel Paragon is linked with the Paragons located at Oak Ridge National Laboratories using GigaNet ATM technology. While also termed a metacomputing environment, the consortium initially targets at running specific multi-site applications (e.g., climate model) in distributed mode.

The well-known Berkeley *NOW* project [33] emphasizes on using networks of workstations mainly for improving virtual memory and file system performance by using the aggregate main memory of the network as a giant cache. Moreover, highly available and scalable file storage is provided by using the redundant arrays of workstation disks, and—of course—the multiple CPUs are used for parallel computing.

### 3 MOL Architecture

We regard a metacomputer as a dynamic entity of interacting modules. Consequently, interface specifications play a central role in the MOL architecture, as illustrated in Figure 1. There exist three general module classes:

1. programming environments,
2. resource management & access systems,
3. supporting tools.

(1) *Programming environments* provide mechanisms for communicating between threads and they allow the use of specific system resources. MPI and

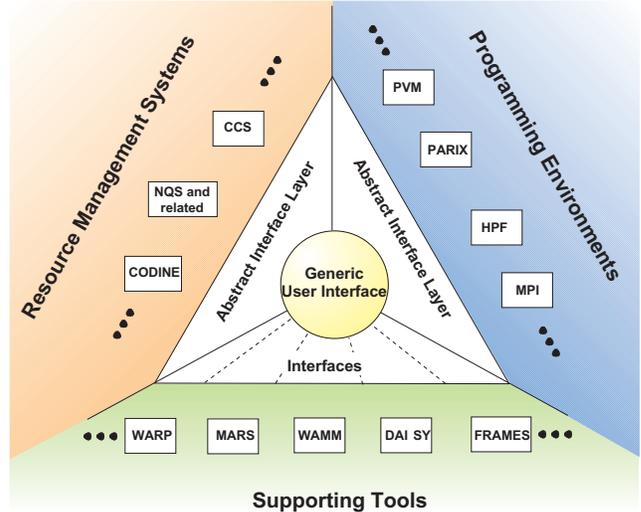


Figure 1: MOL architecture

PVM, for example, are popular programming environments supported by MOL. More important, MOL also supports the communication and process management of *heterogeneous* applications comprising software using different programming environments. The MOL library “PLUS” makes it possible, for example, to establish communication links between MPI- and PVM-code that are part of one large application. PLUS is almost transparent to the application, requiring only a few modifications in the source code (see Sec. 5).

(2) *Resource management & access systems* provide services for starting an application and for controlling its execution until completion. Currently there exists a large number of different resource management systems, each with its specific advantage [3]. Codine, for example, is well suited for managing (parallel) jobs on workstation clusters, while NQS is specialized on serving HPC systems in batch mode. Because the specific advantages of these systems supplement each other, we have developed an *abstract interface layer* with a high-level resource description language that provides a unified access to several (possibly overlapping) resource management systems.

(3) *Supporting tools* provide optional services to metacomputer applications. Currently, the MOL toolset includes software modules for dynamic task migration (MARS), for easy program development by using programming templates (FRAMES), and also data libraries for virtual shared memory and load balancing (DAISY). The wide-area metacomputer manager WAMM plays a special role in MOL: On the one hand, it may be used just as a resource management system for assigning tasks to resources, and on the other hand,

it also provides automatic source code compilation and cleanup after task execution.

## 4 Interface Layers

As described, MOL facilitates the interaction between different resource management systems and different programming environments. However, this is not enough: A metacomputer application may also require some interaction between these two groups. As an example, consider a PVM application running on a workstation cluster that is about to spawn a process on a parallel computer. For this purpose, the application makes a call to the parallel system's management software in order to allocate the necessary resources. Likewise, a metacomputer application may need a mechanism to inform the communication environment about the temporary location of the participating processes.

In addition, tools are needed for efficient task mapping, load balancing, performance prediction, program development, etc. There is a wealth of supporting tools available to cover most aspects of metacomputing. However, as these tools are typically very complex, it is hard to adapt their interfaces for our specific needs. In the MOL framework shown in Figure 1, the supporting tools only need to interact with two abstract interface layers rather than with all systems below. Thus, new software must be integrated only once, and updates to new releases affect only the tool in question.

### 4.1 MOL User Interface

Metacomputing services will be typically accessed via Internet or Intranet world-wide-web browsers, such as the Netscape Navigator or the Microsoft Explorer. MOL provides a generic graphical user interface (GUI) based on HTML and Java for interactive use and for submitting batch jobs. At the top level, the GUI displays general information on status and availability of the system services. Users may select from a number of alternatives by push-down buttons. In doing so, context sensitive windows are opened to ask for context specific parameters required by the requested services. Figure 2 shows the MOL window used for submitting interactive or batch jobs.

This concept is called 'interface hosting': The generic interface acts as a host for the sub-interfaces of the underlying modules. Interface hosting gives the user control over the complete system without the need to change the environment when the user wants to deal with another feature or service of the metacomputer.

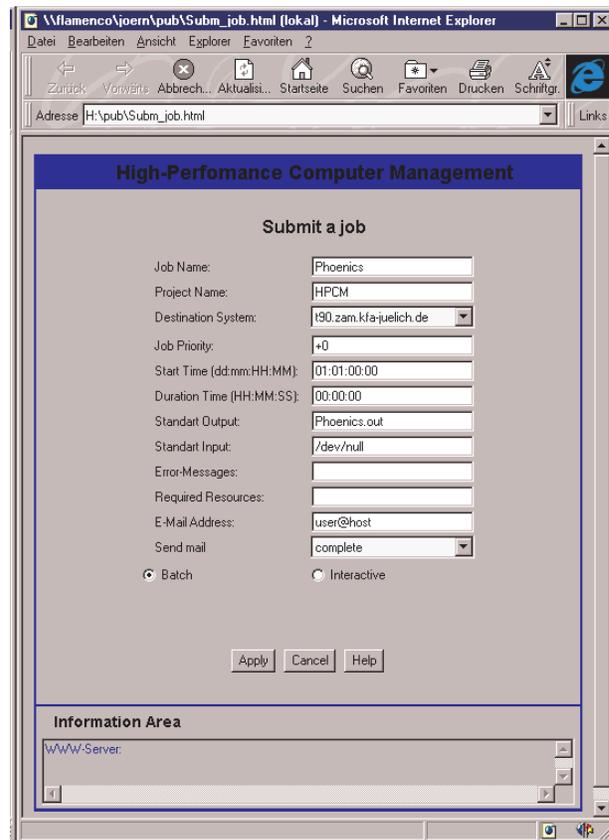


Figure 2: Prototype GUI of Paragon/Parsytec metacomputer

The different skills of users are met by a Java-based graphical interface and a shell-based command oriented interface. In addition, an API library allows applications to directly interact with the metacomputer services. All three interfaces,

- the graphical user interface (Java)
- the shell-based command line interface
- the API library

provide the same functions.

Figure 2 illustrates the graphical user interface used to submit a job to a Parsytec/GC and a Paragon located in Paderborn and Jülich. The two machines are installed 300 kilometers apart. They are interconnected via the 34Mbps German Research WAN. Note that the same interface is used for submitting batch jobs as well as for starting interactive sessions.

A later version of the interface will include a graph editor, allowing the user to describe more complex interconnection topologies and hierarchical structures.

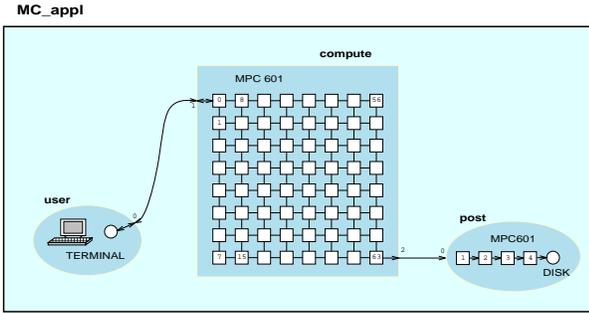


Figure 3: Simple heterogeneous computing example

When the actual system configuration has been determined by the resource scheduler, the resulting topology will be presented to the user by a graph-drawing tool [32].

## 4.2 A MOL User Session

In this section, we describe the specification and execution of an example user application that is executed in the MOL environment. Suppose that a large grid structured application shall be run on a massively parallel system with 64 PowerPC 601 nodes, each of them having at least 16 MB of main memory. As shown in Figure 3, the result of this computation shall be post-processed by another application with a pipeline structure. Depending on the available resources, the resource management system may decide to collapse the pipeline structure to be run on a single processor, or to run the grid and the pipeline on the same machine. For interactive control, the user terminal is linked to the first node of the grid part of the application. For the concrete resource description used to specify this metacomputer scenario see Figure 6 below.

Within MOL, such resource specifications are generated by tools that are integrated into the GUI. When submitting a resource request, the request data is translated into the internal format used by the abstract resource management (RM) interface layer. The result is then handed over to the scheduler and configurator. The RM interface layer schedules this request and chooses a time slot when all resources are available. It contacts the target management systems and takes care that all three programs will be started at the same time. Furthermore, it generates a description file informing the abstract programming environment interface where the programs will be run and which addresses to be used for establishing the first communication links. At program run time, the programming environment interface layer establishes the

interconnection between the application domains and takes care of necessary data conversion.

Though this example is not yet completely realized, it gives an idea on how the modules would interact with each other by means of abstract layers and how the results are presented to the user by the generic interface.

## 5 Programming Environments Layer

In contrast to many other heterogeneous computing environments, MOL is not restricted to a specific programming environment. Applications may use any programming model supported by the underlying hardware platforms. This could be either a vendor supplied library or a standard programming model such as PVM and MPI.

However, many programming models are homogeneous, that is, applications can only be executed on the system architecture they have been compiled (linked) for. While there also exist some heterogeneous programming libraries, they usually incur significant performance losses due to the high level of abstraction. Portable environments are typically implemented on top of the vendor's libraries, making them an order of magnitude slower than the vendor supplied environments. Clearly, users are not willing to accept any slow-downs in their applications, just because the application might need some infrequent communication with some remote system. This problem is addressed by PLUS.

### 5.1 PLUS – A Linkage Between Programming Models

*PLUS* stands for *Programming Environment Linkage by Universal Software Interfaces* [12]. It provides a lightweight interface between different programming environments which are only used for communicating with remote hosts. The efficiency of the native environment remains untouched.

The interfaces are embedded into the various environments. A PVM application, for example, uses ordinary PVM-IDs for addressing non-PVM processes. Consequently, a PARIX process reachable via PLUS is represented within a PVM application by an ordinary PVM-ID. Of course, PLUS takes care that these 'pseudo PVM-IDs' do not conflict with the real PVM-IDs.

Furthermore, PLUS allows to create processes by overlaying existing routines (PVM) or by extending the available function set with its own procedure (in the case of MPI). Thereby, PLUS adds a dynamic process model to programming environments like MPI1,

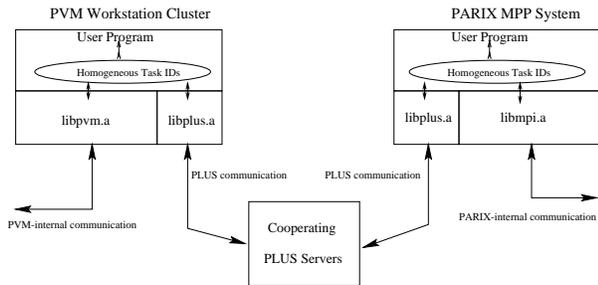


Figure 4: PLUS architecture

which do not provide such facilities. PLUS has no internal resource management functionality, but it forwards the requests to the abstract resource management layer, which offers more powerful methods than could be integrated into PLUS.

With this concept, PLUS allows program developers to integrate existing codes into the metacomputer environment, thereby lowering barriers in the use of metacomputers.

**PLUS Architecture.** Fig. 4 depicts the architecture of the abstract interface layer for programming environments developed in the PLUS project. Note that the regular PVM communication is not affected by PLUS. Only when accessing a PVM-ID that actually represents an external process (a PARIX process in this example), the corresponding PLUS routine is invoked. This routine performs the requested communication via the fastest available protocol between the PVM cluster and the PARIX system. Usually, communication will be done via UDP.

PLUS libraries are linked to each other via one or more PLUS server. Since most of the PLUS code is contained in these servers, the link library could be kept rather small. The servers manage the translation of different data representations, the message routing along the fastest network links, the dynamic creation of new processes, and much more. The number of active PLUS servers and their configuration may change dynamically at run time according to the needs of the user's application.

**PLUS Performance.** On a 34 Mb/s WAN interconnection, the PLUS communication has been shown to be even faster than TCP [12]. This is because the UDP based communication protocol of PLUS builds a virtual channel on which the messages are multiplexed. A sliding window technique allows to assemble and re-order the acknowledgements in the correct sequence.

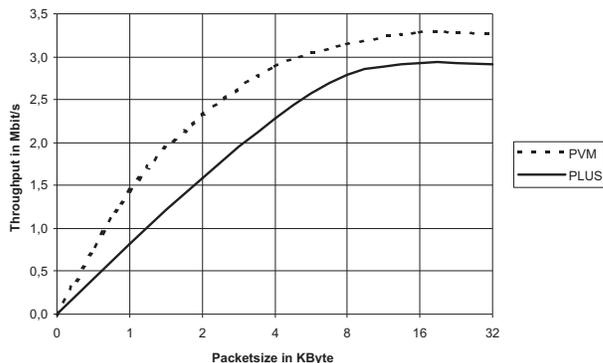


Figure 5: PLUS versus PVM on a 10Mb/s LAN

Figure 5 shows a less favorable case where the PLUS communication is outperformed by PVM. A parallel Parsytec CC system under AIX has been interconnected to a SUN-20 via 10Mb/s Ethernet. In both runs, the same PVM code has been used: once communicating via the internal PVM protocol, and the other time via the PLUS protocol. Here, PLUS is about 10 to 20% slower, because the PLUS communication needs two Internet hops, one hop from the CC to the PLUS daemon, and another from the daemon to the target SUN. The PVM tasks, in contrast, need only one Internet hop for communicating between the two corresponding PVM daemons. Moreover, since the PLUS communication libraries are designed in an open and extensible way, they do not contain routing information.

Note, that the example shows a worst case, because PLUS would only be used to communicate between *different* programming environments. Any internal communication, e.g. within PVM, is not affected by PLUS.

**Extending PLUS.** The current version of PLUS supports PVM, MPI and PARIX, where the latter is available on Parsytec systems only.

The PLUS library consists of two parts: a generic, protocol-independent module, and a translation module. The translation module contains a small set of abstract communication routines for the specific protocols. These routines are quite small, typically comprising less than one hundred lines of code.

New programming environments can be integrated to PLUS by implementing a new translation module. This allows applications to communicate with any other programming model for which translation modules are available in PLUS.

## 6 Resource Management Layer

Resource management is a central task in meta-computing environments, permitting oversubscribed resources to be fairly and efficiently shared. Historically, supercomputer centers have been using batch queuing systems such as NQS [28] for managing their machines and for scheduling the available computing time. Distributed memory systems employ more complex resource scheduling mechanisms, because a large number of constraints must be considered in the execution of parallel applications. As an example, interactive applications may compete with batch jobs, and requests for the use of non-timeshared components or for special I/O facilities may delay other jobs.

On a metacomputer, the scheduler is responsible for assigning appropriate resources to parallel applications. Scheduling is done by the resource management system that allocates a particular CPU at a particular time instance for a given application, and by the operating system running on a certain compute node. In parallel environments, we distinguish two levels of scheduling:

- At the *task level*, the scheduling of tasks and threads (possibly of different jobs) is performed by the operating system on the single MPP nodes. The scheduling strategy may be uncoordinated, as in the traditional time-sharing systems, or it can be synchronized, as in gang-scheduling. The latter, however, is only available on a few parallel systems.
- At the *job level*, the scheduling is usually architecture independent. A request may specify the CPU type, memory requirements, I/O facilities, software requirements, special interconnection structures, required occupation time and some attributes for distinguishing batch from interactive jobs. Here, the request scheduler is responsible for re-ordering and assigning the submitted requests to the most appropriate machines.

With few exceptions [9], the existing schedulers were originally designed for scheduling serial jobs [27, 3]. As a consequence, many of them do not obey the 'concept' of parallel programs [37]. Typically, a stater (master) process is launched that is responsible for starting the rest of the parallel program. Thus, the management has limited knowledge about the distributed structure of the application. When the master process dies unexpectedly (or has been exempted because the granted time is expired) the rest of the

parallel program is still existent. Such orphaned processes can cause serious server problems, and in case of MPP systems (like the SP2) they can even lock parts of the machine.

With its open architecture, MOL is not restricted to the usage of specific management systems. Rather, we distinguish three types of management systems, each of them specialized to a certain administration strategy and usage profile:

The first group of applications comprise *sequential and client-server programs*. The management software packages of this group have emerged from the traditional vector-computing domain. Examples are Condor, Codine, DQS and LSF.

The second group contains resource management systems tailored to the needs of *parallel applications*. Some of them have their roots in the NQS development (like PBS), while others were developed to overcome problems with existing vendor solutions (like EASY) or do focus on the transparent access to partitionable MPP systems (like CCS described below).

The last group consists of resource management systems for *multi-site applications*, exploiting the whole power of a metacomputing environment. Currently, only few multi-site applications exists, and the development of corresponding resource management systems is still in its infancy. But with increasing network performance such applications and their need for a uniform and optimizing management layer is gaining importance.

### 6.1 Computing Center Software CCS

CCS [15] provides transparent access to a pool of massively parallel computers with different architectures [35]. Today, parallel machines ranging from 4 to 1024 nodes are managed by CCS. CCS provides user authorization, accounting, and for scheduling arbitrary mixtures of interactive and batch jobs [23]. Furthermore, it features an automatic reservation system and allows to re-connect to a parallel application in the case of a breakdown in the WAN connection—an important feature for remote access to a metacomputer.

#### Specification Language for Resource Requests.

Current management systems use a variety of command-line (or batch script) options at the user interface, and hundreds of environment variables (or configuration files) at the operator interface. Extending such concepts to a metacomputing environment can result in a nightmare for users and operators.

Clearly, we need a general resource description language equipped with a powerful generation and anima-

```

INCLUDE <default_defs.rdl>                                -- default definitions and constant declarations

DECLARATION
BEGIN UNIT MC_appl;

  DECLARATION
  BEGIN SECTION main;                                    -- main computation on grid
  EXCLUSIVE;                                           -- use separate processors for each item
  DECLARATION
    FOR i=0 TO (main_x * main_y - 1) DO
      { PROC i; compute = pbl_size; CPU = MPC601; MEMORY = 16; }; OD
  CONNECTION
    FOR i=0 TO (main_x - 1) DO
      FOR j = i * main_y TO (i * main_y + main_y-2) DO
        PROC j LINK 0 <=> PROC j+1 LINK 2; OD OD
      FOR i=0 TO (main_y - 1) DO
        FOR j=0 TO (main_x - 2) DO
          PROC (j * main_y + i) LINK 1 <=> PROC (main_y * (j + 1) + i) LINK 3; OD OD

    ASSIGN LINK 1 <=> PROC 0 LINK 3;                    -- from user terminal
    ASSIGN LINK 2 <== PROC (main_x * main_y-1) LINK 1; -- to postprocessing
  END SECTION

  BEGIN SECTION post;                                   -- pipelined post-processing
  SHARED;                                              -- the items of this section may be run on a single node
  DECLARATION
    FOR i=1 TO post_len DO
      { PROC i; filter = post; CPU = MPC601; }; OD
      { PORT Ausgabe; DISK; };
  CONNECTION
    FOR i=1 TO (post_len - 1) DO
      i LINK 0 ==> i + 1 LINK 1; OD
    PROC post_len LINK 0 ==> PORT Ausgabe LINK 0;
    ASSIGN PROC 1 LINK 1 <== LINK 0; -- link to higher level
  END SECTION

  BEGIN SECTION user;                                  -- user control section
  DECLARATION
    { PORT IO; TERMINAL; };
  CONNECTION
    ASSIGN IO LINK 0 <=> LINK 0;
  END SECTION

  CONNECTION                                           -- connecting the modules
    user LINK 0 <=> main LINK 1;
    main LINK 2 ==> post LINK 0;
END UNIT -- MC_appl

```

Figure 6: Specification of the system shown in Figure 3 using the Resource Description Language RDL

tion tool. Metacomputer users and metacomputer administrators should not have to deal directly with this language but only with a high-level interface. Analogously to the popular *postscript* language used for the device-independent representation of documents, a metacomputer resource description language is a means to specify the resources needed for a metacomputer application. Administrators should be able to use the same language for specifying the available resources in the virtual machine room.

In a broad sense, resource representations must be specified on three different abstraction levels: The *vendor-level* for describing the internal structure of a machine. The *operator-level*, which is the metacomputer itself, for describing the interconnections of the machines within the network and their general properties. And the *user-level*, for specifying a logical topology to be configured for a given application.

Within MOL, we use the *Resource Description Language RDL* [4], that has been developed as part of the CCS resource management software [35]. RDL is used

- at the administrator's level for describing type and topology of the participating metacomputer components, and
- at the user's level for specifying the required system configuration for a given application.

Figure 6 shows an RDL specification for the example discussed in Figure 3. It is the task of the resource scheduler to determine an optimal mapping between the two specifications: The specification of the application structure on the one hand, and the specification of the system components on the other hand. Better mapping results are obtained when the user requests are less specific, i.e., when classes of resources are specified instead of specific computers.



Figure 7: Snapshot of WAMM user interface

## 7 Supporting Tools

The MOL toolset contains a number of useful tools for launching and executing distributed applications on the metacomputer. We first describe the “wide area metacomputer manager” WAMM, which provides resource management functionality as well as automatic source code compilation and cleanup after task execution.

Another class of tools allows for dynamic task migration at execution time (MARS), which makes use of the performance prediction tool WARP.

Data libraries for virtual shared memory and load balancing (DAISY) provide a more abstract programming level and program templates (FRAMES) facilitate the design of efficient parallel applications, even for non-experts.

### 7.1 Graphical Interface WAMM

The *Wide Area Metacomputer Manager* WAMM [43, 5, 6] supports the user in the management of the computing nodes that take part in a parallel computation. It controls the virtual machine configuration, issues remote commands and remote source code compilations, and it provides task management. While earlier releases of WAMM [5, 6] were limited to systems running PVM only, with the PLUS library it is now possible to use WAMM on arbitrary systems [7].

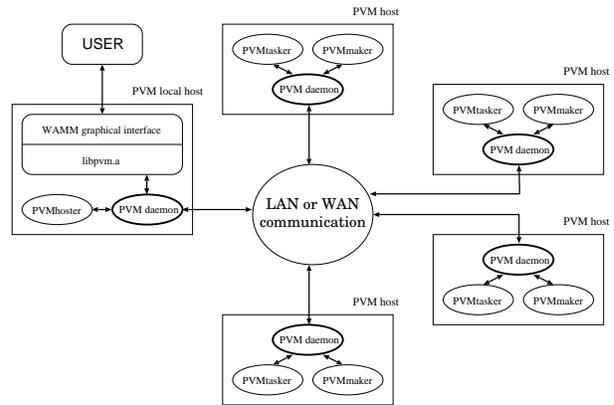


Figure 8: WAMM architecture

WAMM can be seen as a mediator between the top user access level and the local resource management. It simplifies the use of a metacomputer by adopting a “geographical” view, where the hosts are grouped in tree structured sub-networks (LANs, MANs or WANs). Each item in the tree is shown in an OSF/Motif window, using geographical maps for networks and icons for hosts as shown in Figure 7. The user can move through the tree and explore the resources by selecting push buttons on the maps. It is possible to zoom from a wide-area map to a single host of a particular site. A traditional list with Internet host addresses is also available.

**Metacomputer Configuration.** The metacomputer can be configured by writing a configuration file which contains the description of the nodes that make up the metacomputer, i.e. all the machines that users can access. This file will be read by WAMM at startup time.

**Application Development.** The development and execution of an application requires some preparatory operations such as source code editing, remote compilation and execution. In WAMM, programmers develop their code on a local machine. For remote compilation, they only have to select hosts where they want to do the compilation and to issue a single **Make** command from the popup menu. In a dialog box the local directories containing the source files and corresponding **Makefiles** can be specified along with the necessary parameters.

WAMM supports remote compilation by grouping all source files into a single, compressed **tar** file. A

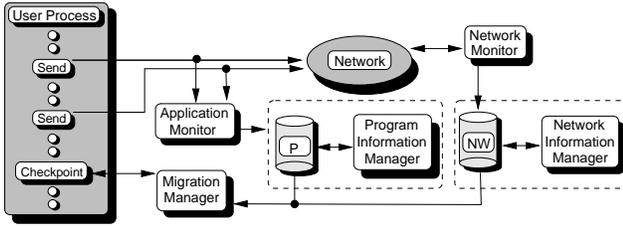


Figure 9: MARS System Architecture

**PVMMaker** task, which deals with the remote compilation, is spawned on each participating node and the compressed tar file is sent to all these tasks. The rest of the work is carried out in parallel by the **PVMMakers**. Each **PVMMaker** receives the compressed file, extracts the sources in a temporary working directory, and executes the **make** command. WAMM is notified about the operations that have been executed, and the result is displayed in a control window to show the user the status of the compilation.

**Tasks Execution and Control.** Application are started by selecting the **Spawn** pushbutton from the **Apps** popup menu. A dialog box is opened for the user to insert parameters, such as number of copies, command line arguments, etc. The output of the processes is displayed in separate windows and/or saved in files. When the output windows are open, new messages from the tasks are shown immediately (Fig. 7). A **Tasks** control window can be opened to control some status information on all the PVM tasks being executed in the virtual machine.

## 7.2 MARS Task Migrator

The *Metacomputer Adaptive Runtime System MARS* [24] is a software module for the transparent migration of tasks during runtime. Task migrations may become necessary when single compute nodes or sub-networks exhibit changing load due to concurrent use by other applications. MARS uses previously acquired knowledge about a program's runtime behavior to improve its task migration strategy. The knowledge is collected during execution time without increasing the overall execution time significantly. The core idea is to keep all information gathered in a previous execution of the same program and to use it as a basis for the next run. By combining information from several runs, it is possible to find regularities in the characteristic program behavior as well as in the network profile. Future migration decisions are likely to benefit by the acquired information.

The MARS runtime system comprises two types of instances (Fig. 9): *Monitors* for gathering statistical data on the CPU work-load, the network performance and the applications' communication behavior, and *Managers* for exploiting the data for computing an improved task-to-processor mapping and task migration strategy.

As MARS is designed for heterogeneous metacomputers, we cannot simply migrate machine-dependent core images. Instead, the application code must be modified by a preprocessor to include calls to the runtime system at certain points where task migration is allowed. The user's object code is linked to the MARS runtime library which notifies an *Application Monitor* and a *Network Monitor* each time a **send** or **receive** operation is executed.

The Network Monitor collects long-term statistics on the network load. The Application Monitor monitors the communication patterns of the single applications and builds a task dependency graph for each execution. Dependency graphs from successive execution runs are consolidated by a *Program Information Manager*. The resulting information is used by the *Migration Manager* to decide about task migrations whenever a checkpoint is reached in the application.

In summary, two kinds of data are maintained: application specific information (in dependency graphs) and system specific information (in system tables) for predicting the long-term performance of the network and CPU work-load. Further information on MARS can be found in [24].

## 7.3 Runtime Predictor WARP

*WARP* is a Workload Analyzer and Runtime Predictor [39, 40]. Performance prediction tools provide performance data to compilers, programmers and system architects to assist in the design of more efficient implementations. While being an important aspect of high-performance computing, there exist only few projects that address performance prediction for clusters of workstations and no project targets the metacomputing scenery.

WARP is used within MOL to advise the resource management systems for better hardware exploitation by maximizing the through-put or by minimizing the response time. More specifically, the performance prediction figures of WARP provide valuable input for the initial mapping and dynamical task migration performed by MARS. The WARP system [39, 40] includes modules for

- compiling resource descriptions into a suitable internal representation,

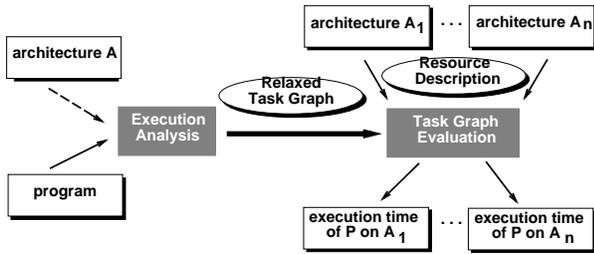


Figure 10: WARP architecture

- analyzing resource requests of parallel programs, and
- predicting the execution time of parallel programs.

**Architectural Model.** In WARP, a parallel machine is described by a hierarchical graph. The nodes in the graph denote processors or machines with local memory. The edges denote interconnections between the machines. The nodes are weighted with the relative execution speed of the corresponding machines, which have been measured by small benchmark programs or routines for testing the functional units of the CPU (floating-point pipelines, load & store operations in the memory hierarchy, etc.). The edges are weighted with the latency and bandwidth of the corresponding communication performance between the two nodes, which can be either a local mechanism (e.g., shared memory) or a network communication. A monitoring tool in WARP detects the basic load of time-sharing systems, that is used to model the statistical load reducing the potential performance of the machine or interconnection.

**Task Graph Construction.** WARP models the execution of a parallel program by a task graph consisting of a set of sequential blocks with corresponding interdependencies. The nodes and edges of a task graph are weighted with their computation and communication loads. Task graphs are constructed either by static program analysis or by executing the code on a reference system. In the latter case, the load factors are reconstructed from the execution time of the sequential blocks and communications.

Clearly, the task graphs – representing execution traces – must not be unique. In cases where the control

flow depends on the execution sequence of the sequential blocks or on the communication pattern, stochastic graphs are used. Moreover, detailed event tracing can result in extremely large task graphs, depending on the size of the system. Fortunately, most parallel applications are written in SPMD or data parallel mode, executing the same code on all processors with local, data-dependent branches. Only a few equivalence classes of processor behavior are modeled by WARP, with statistical clustering used as a standard technique for identifying data equivalence classes. Also regular structures derived by loops and subroutines are used for a task graph relaxation.

**Task Graph Evaluation.** Task graphs are evaluated for predicting the execution time under modified task allocation schemes and/or changed resources. The task graph evaluation is done by a discrete event simulator which constructs the behavior of the parallel program running on a hardware with a given resource description. Resource contention is modeled by queuing models. The execution times and communication times are then adjusted to the relative speed of the participating resource and the expected contention.

#### 7.4 Data Management Library DAISY

The data management library *DAISY* (Fig. 11) comprises tools for the simulation of shared memory (DIVA) and for load balancing (VDS) in a single comprehensive library. A beta release of DAISY is available for Parsytec’s PARIX and PowerMPI programming models. With the improved thread support of MPI-2, DAISY will also become available on workstation clusters.

**Load Balancing with VDS.** The virtual data space tool *VDS* simulates a global data space for structured objects stored in distributed heaps, stacks, and other abstract data types. The work packets are spread over the distributed processors as evenly as possible with respect to the incurred balancing overhead [19]. Objects are differentiated by their corresponding class, depicted in Figure 11 by their different shape. Depending on the object type, one of three distribution strategies is used:

- *Normal* objects are weighted by a load measure given by the expense of processing the object. VDS attempts to distribute the object such that all processors have about the same load. A processor’s load is defined as the sum of the load-weights of all placed objects.

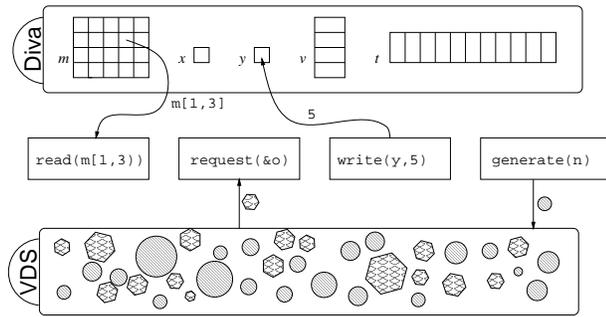


Figure 11: : The two *DAISY* tools: Distributed shared memory (Diva) and the load-balancing layer VDS.

- In some applications, such as best-first branch-and-bound, the execution time is not only affected by the number of objects, but also by the order in which the objects are processed. In addition to the above described quantitative load balancing, some form of qualitative load balancing is performed to ensure that all processors are working on promising objects. VDS provides *qualitative load balancing* by means of the weighted objects. Besides their load-weight, these objects possess a quality tag used to select the next suitable object from several alternatives.
- The third kind of object, *thread objects*, provide an easy and intuitive way to model multi-threaded computations as done in CILK [10]. In this computational model, threads may send messages (results) to their parents. With later versions it will be possible to send data across more than one generation (e.g. to grandparents) [21].

In the current VDS release, we implemented a work-stealing method [11] for thread objects as well as diffusive load balancing for normal and weighted objects.

**Shared Memory Simulation with DIVA.** The distributed variables library *DIVA* provides functions for simulating shared memory on distributed systems. The core idea is to provide an access mechanism to distributed variables rather than to memory pages or single memory cells. The variables can be created and released at runtime. Once a global variable is created, each participating processor in the system has access to it.

For latency hiding, reads and writes can be performed in two separate function calls. The first call initiates the variable access, and the second call waits

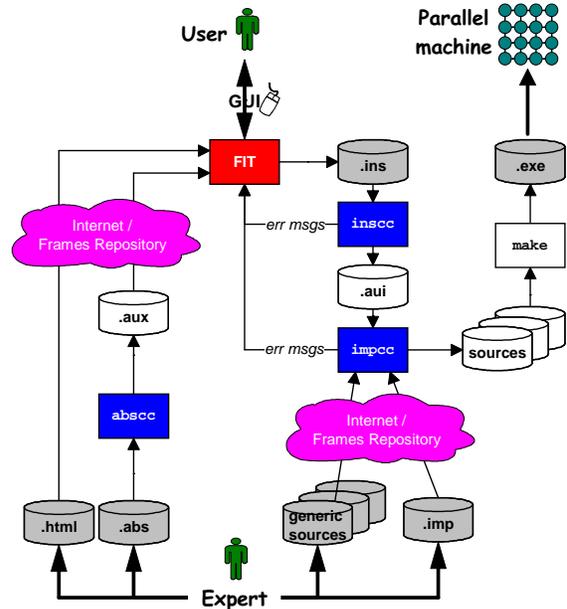


Figure 12: The frame model

for its completion. The time between initiation and completion of a variable access can be hidden by other local instructions or variable accesses.

## 7.5 Programming Frames

Programming frames facilitate the development of efficient parallel code for distributed memory systems. Programming frames are intended to be used by non-experts, who are either unfamiliar with parallel systems or unwilling to cope with new machines, environments and languages. Several projects [16, 8, 17, 18] have been initiated to develop new and more sophisticated ways for supporting the programming of distributed memory systems via libraries of basic algorithms, data structures and programming frameworks (templates). Like LEDA for the sequential case [30], each of these approaches provides non-experts with tools to program and exploit parallel machines efficiently.

Our frames are like black boxes with problem dependent holes. The basic idea is to comprise expert knowledge about the problem and its parallelization into the black box and to let the users specify the holes, i.e. the parts that are different at each instantiation. The black boxes are either constructed using efficient basic primitives, standard parallel data

types, communication schemes, load balancing and mapping facilities, or they are derived from well optimized, complete applications. In any case, frames contain efficient state-of-the-art techniques focusing on reusability and portability – both important aspects in metacomputing. This will save software development costs and improve the reliability of the target code.

Figure 12 depicts our frame model. Each generated target executable is built from three different specifications. The *abstract specification* defines the parameters of the problem (the holes in the black box). It remains the same for all instances of that frame, and is generally given by an expert. Furthermore, the abstract specification is used to generate a graphical user interface (GUI) and for the type consistency check in the instance level.

At the *instance level*, values are assigned to the parameters specified in the abstract level. New problem instances are generated by the user by giving new instance specifications via the GUI.

The implementation level contains a number of implemented sources, e.g. for MPI or PVM, with respect to a given abstract frame. An *implementation specification* consists of a list of source files and rules for modifying those sources to get new ones that comply with the values given in the instance level. The rules are best described as replacements and generations. New frames can also be composed by modifying and/or composing existing basic frames.

Three major tools are used to process the frame specifications. The first checks the abstract specification. The second one checks the instance specification against the abstract specification. The build tool, finally, generates the target executables taking the specifications for both an instance and an implementation. Our preliminary versions of these tools and the GUI are portable across systems with POSIX compliant C compilers. As GUIs are considered vital for the acceptance of the programming frames, our implementation is based on the graphical TCL/TK toolkit. The interface reads the abstract specification and prompts the user for the frame parameters. The output is an instance specification according to our model. A Java interface to the frames will be available in the near future.

## 8 Summary

We have presented an open metacomputer environment that has been designed and implemented in a collaborative effort in the Metacomputer Online (MOL) initiative. Due to the diversity in the participating hard- and software on the one hand, and due to the

heterogeneous spectrum of potential users on the other hand, we believe that a metacomputer cannot be regarded as a closed entity. It should rather be designed in an open, extensible manner that allows for continuous adjustment to meet the user's demands by a dynamically changing HW/SW environment.

With the MOL framework, we have linked existing software packages by generic, extensible interface layers, allowing future updates and inclusion of new soft- and hardware. There are three general classes of metacomputer modules:

- programming environments (e.g., PVM, MPI, PARIX, and the PLUS linkage module),
- resource management & access systems (Codine, NQS, PBS, CCS),
- supporting tools (GUIs, task migrator MARS, programming frames, data library DAISY, WAMM, WARP performance predictor,...).

All of these modules exist. Linked by appropriate generic interfaces, the modules became an integral part of the MOL environment. From the hardware perspective, MOL currently supports geographically distributed high-performance systems like Parsytec GC, Intel Paragon, and UNIX workstation clusters that are run in a dedicated compute cluster mode.

As a positive side-effect, the collaboration within the MOL group has resulted in a considerable amount of (originally unexpected) synergy. Separate projects, that have been started as disjoint research work, now fit together in a new framework. As an example, the task migration manager MARS derives better migration decisions when being combined with the performance predictor WARP. An even more striking example is the linkage of WAMM with PLUS [7]. Previously, the WAMM metacomputer manager was limited to PVM only. The PLUS library now extends the application of WAMM to a much larger base of platforms.

## Acknowledgements

MOL is the collaborative effort of many individuals. We are indebted to the members of the Metacomputer Online Group, the members of the Northrhine-Westphalian Initiative Metacomputing, and to the members of ALCOM-IT, especially Jordi Petit i Silvestre (UPC).

## References

- [1] D. Abramson, R. Sasic, J. Giddy, B. Hall. *Nimrod: A tool for performing parameterized simulations using distributed workstations*. 4th IEEE Symp. High-Perf. Distr. Comp. (August 1995).
- [2] J.N.C. Áraabe, A.B.B. Lowekamp, E. Seligman, M. Starkey, P. Stephan. *Dome: Parallel programming environment in a heterogeneous multi-user environment*. Supercomputing 1995.
- [3] M.A Baker, G.C. Fox, H.W. Yau. *Cluster computing review*. Techn. Report, Syracuse Univ., Nov. 1995.
- [4] B. Bauer, F. Ramme. *A general purpose Resource Description Language*. Reihe Informatik aktuell, Hrsg R. Grebe, M. Baumann, Parallel Datenverarbeitung mit dem Transputer, Springer-Verlag, (Berlin), 1991, 68-75.
- [5] R. Baraglia, G. Faieta, M. Formica, D. Laforenza. *WAMM: A Visual Interface for Managing Metacomputers*. EuroPVM'95, Ecole Normale Supérieure de Lyon, Lyon, France, September 14-15, 1995, 137-142.
- [6] R. Baraglia, G. Faieta, M. Formica, D. Laforenza. *Experiences with a Wide Area Network Metacomputing Management Tool using IBM SP-2 Parallel Systems*. Concurrency: Practice and Experience, John Wiley & Sons, Ltd., Vol.8, 1996, in press.
- [7] R. Baraglia, D. Laforenza. *WAMM integration into the MOL Project*. CNUCE Inst. of the Italian Nat. Research Council, Pisa, Internal Rep., 1996.
- [8] R. Barret, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst. *TEMPLATES for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Tech. Rep., CS-Dept., Univ. of Tennessee, 1993.
- [9] F. Berman, R. Wolski, S. Figueira, J. Schopf, G. Shao. *Application-level scheduling on distributed heterogeneous networks*. Tech. Rep., Univ. California, San Diego. <http://www-cse.ucsd.edu/~groups/hpcl/apples>.
- [10] R.D. Blumhofs, C.F. Joerg, B.C. Kuszmaul, C.E. Leiserson, K.H. Randall, Y. Zhou. *Cilk: An Efficient Multithreaded Runtime System*. Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, July 19-21, 1995, Santa Barbara, California, 207-216.
- [11] R.D. Blumhofs, C.E. Leiserson. *Scheduling Multithreaded Computations by Work Stealing*. Proc. of the 36th Ann. Symposium on Foundations of Computer Science (FOCS '95), 356-368, 1995.
- [12] M. Brune, J. Gehring, A. Reinefeld. *A Lightweight Communication Interface Between Parallel Programming Environments*. HPCN'97, Springer LNCS.
- [13] G. D. Burns, R. B. Daoud, J. .R. Vaigl. *LAM: An Open Cluster Environment for MPI*. Supercomputing Symposium '94, Toronto, Canada, June 1994
- [14] N. Carriero, E. Freeman, D. Gelernter, D. Kaminisky. *Adaptive Parallelism and Piranha*. IEEE Computer 28,1 (1995), 40-49.
- [15] *Computing Center Software CCS*. Paderborn Center for Parallel Computing. <http://www.uni-paderborn.de/pc2/projects/ccs>
- [16] M. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. PhD, Research Monographs in Par. and Distr. Computing, MIT Press.
- [17] M. Danelutto, R. Di Meglio, S. Orlando, S. Pelagatti, M. Vanneschi. *A Methodology for the Development and the Support of Massively Parallel Programs*. J. on Future Generation Computer Systems (FCGS), Vol. 8, 1992.
- [18] J. Darlington, A.J. Field, P.G. Harrison, P.H.J. Kelly, D.W.N. Sharp, Q. Wu. *Parallel Programming Using Skeleton Functions*. Proc. of Par. Arch. and Lang. Europe (PARLE '93), Lecture Notes in Computer Science No. 694, Springer-Verlag, 1993.
- [19] T. Decker, R. Diekmann, R. Lüling, B. Monien. *Towards Developing Universal Dynamic Mapping Algorithms*. Proc. of the 7th IEEE Symposium on Parallel and Distributed Processing, SPDP'95, 1995, 456-459.
- [20] J.J. Dongarra, S.W. Otto, M. Snir, D. Walker. *A message passing standard for MPP and Workstations*. CACM 39,7(July 1996), 84-90.

- [21] P. Fatourou, P. Spirakis. *Scheduling Algorithms for Strict Multithreaded Computations*. Proc. of the 7th Annual International Symposium on Algorithms and Computation (ISAAC '96), 1996, to appear.
- [22] I. Foster. *High-performance distributed computing: The I-Way experiment and beyond*. Proc. of the EURO-PAR'96, Lyon, 1996, Springer LNCS 1124, 3-10.
- [23] J. Gehring, F. Ramme. *Architecture-Independent Request-Scheduling with Tight Waiting-Time Estimations*, IPPS Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 1162, 1996.
- [24] J. Gehring, A. Reinefeld. *MARS - A Framework for Minimizing the Job Execution Time in a Metacomputing Environment*. Future Generation Computer Systems (FGCS), Elsevier Science B. V., Spring 1996.
- [25] A. Grimshaw, J.B. Weissman, E.A. West, E.C. Loyot. *Metasystems: An approach combining parallel processing and heterogeneous distributed computing systems*. J. Par. Distr. Comp. 21 (1994), 257-270.
- [26] L. V. Kale, S. Krishnan. *CHARM++: A Portable Concurrent Object Oriented System Based On C++*. Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA), September 1993
- [27] J.A. Kaplan, M.L. Nelson. *A Comparison of Queuing, Cluster and Distributed Computing Systems*. NASA Technical Memo, June 1994.
- [28] B.A. Kinsbury. *The Network Queuing System*. Cosmic Software, NASA Ames Research Center, 1986.
- [29] M.J. Litzkow, M. Livny. *Condor-A hunter of idle workstations*. Proc. 8th IEEE Int. Conf. Distr. Computing Systems, June 1988, 104-111.
- [30] K. Mehlhorn, S. Näher. *LEDA, a Library of Efficient Data Types and Algorithms*. MFCS 89, LNCS Vol. 379, 88-106
- [31] Metacomputer Online (MOL). <http://www.uni-paderborn.de/pc2/projects/mol/>
- [32] B. Monien, F. Ramme, H. Salmen : *A Parallel Simulated Annealing Algorithm for Generating 3D Layouts of Undirected Graphs*. Proc. of Graph Drawing '95, Springer LNCS, Vol. 1027, pp. 396-408.
- [33] David Patterson et al. *A Case for Networks of Workstations: NOW*. IEEE Micro. <http://now.CS.Berkeley.EDU/Case/case.html>
- [34] Polder. <http://www.wins.uva.nl/projects/polder/>
- [35] F. Ramme, T. Römke, K. Kremer. *A Distributed Computing Center Software for the Efficient Use of Parallel Computer Systems*. HPCN Europe, Springer LNCS 797, Vol. II, 129-136 (1994).
- [36] F. Ramme, K. Kremer. *Scheduling a Metacomputer by an Implicit Voting System*. 3rd IEEE Int. Symposium on High-Performance Distributed Computing, San Francisco, 1994, 106-113.
- [37] W. Saphir, L.A. Tanner, B. Traversat. *Job Management Requirements for NAS Parallel Systems and Clusters*. IPPS Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 949, 319-336, 1995.
- [38] L. Schäfers, C. Scheidler, O. Krämer-Fuhrmann. *Software Engineering for Parallel Systems: The TRAPPER Approach*. 28th Hawaiian International Conference on System Sciences, January 1995, Hawaii, USA
- [39] J. Simon, J.-M. Wierum. *Performance Prediction of Benchmark Programs for Massively Parallel Architecture*. 10th Annual Intl. Conf. on High-Performance Computer HPCS'96, June 1996.
- [40] J. Simon, J.-M. Wierum. *Accurate Performance Prediction for Massively Parallel Systems and its Applications*. Euro-Par'96 Parallel Processing, August 1996, LNCS 1124, 675-688.
- [41] L. Smarr, C.E. Catlett. *Metacomputing*. Communications of the ACM 35,6(1992), 45-52.
- [42] F. Tandary, S.C. Kothari, A. Dixit, E.W. Anderson. *Batrun: Utilizing idle workstations for large-scale computing*. IEEE Parallel and Distr. Techn., Summer 1996, 41-48.
- [43] WAMM - *Wide Area Metacomputer Manager*. Available at <http://miles.cnuce.cnr.it> or via ftp at <ftp.cnr.it> in the directory /pub/wamm.