Pergamon

0097-8485(95)00084-4

Computers Chem. Vol. 20, No. 4, pp. 431–438, 1996
Copyright © 1996 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0097-8485/96 $15.00 + 0.00

PARALLEL PROCESSING OF CHEMICAL INFORMATION IN A LOCAL AREA NETWORK—I. HYDRA: CONCEPT, CONFIGURATION, AND IMPLEMENTATION OF PARALLEL APPLICATIONS

W. J. MELSSEN,* E. P. P. A. DERKS, M. L. M. BECKERS and L. M. C. BUYDENS

Laboratory for Analytical Chemistry, Faculty of Science, Catholic University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands

(Received 19 June 1995; in revised form 31 August 1995)

Abstract—Sophisticated software packages put an increasing demand on computer hardware. In local area networks, computational intensive programs can lower the performance of individual workstations to an unacceptable level. However, utilizing in a coarse grained sense the computing power of all hosts in such networks, offers the potential to achieve considerable improvements in execution speed within reasonable cost limits. Since conventional workstations are not designed to be used in a parallel configuration, the program HYDRA is developed to control and synchronize parallel processing in a local area network. Part I of this paper focuses on the technical aspects of HYDRA, i.e. configuration and implementation. The second and third parts describe two applications of the HYDRA package in the field of chemistry: using parallel genetic algorithms for the conformational analysis of nucleic acids, and parallel cross-validation of artificial neural networks. Copyright © 1996 Elsevier Science Ltd

1. INTRODUCTION

In academic as well as industrial research environments, computational intensive applications have reached the limit of the Von Neuman paradigm of sequential computing. Although current hardware developments yield computer systems which outperform their predecessors by at least an order of magnitude, the hardware requirements for software packages in, e.g. the fields of natural computation and large-scale multivariate statistical analysis, are still not met satisfactorily. However, a substantial increase of computing power can be achieved by parallelizing conventional software packages and distributing the resulting parallel application on multiple processors. Although massive parallel supercomputers are available nowadays, the concept of exploiting the available computing power of a number of conventional workstations in a local area network, offers a low-cost high-performance solution (Dietz *et al.*, 1994, 1995) and is therefore much more appealing provided that such a network of computers is available. This paper introduces an implementation model of so-called *coarse grained* parallel computing, by means of joining the computational power of multiple computers in a network. The feasibility of this type of parallel processing, applied on a large application, has been demonstrated in a previously conducted pilot study (Melssen *et al.*, 1993). Since no specific constraints have to be put on the processing hardware, various computer systems [e.g. SUN Sparc™ workstations and personal computers] having operating systems supporting the Internet Communication Protocol (e.g. Unix, AIX, Ultrix†) can be assigned to co-operate in parallel.

The type and structure of application oriented software packages that can be run in parallel on multiple workstations is described. The means of controlling parallel-operating computers is provided by the software package "HYDRA"‡ which features among others the distribution, synchronization, monitoring of applications, and controlling of the whole ensemble of computers. Moreover, it provides tools which facilitate the development and implementation of parallel extensions of (existing) single-processor based software packages.

HYDRA provides an application developer the tools to subdivide an arbitrary task into smaller subtasks, hereafter referred to as *instances*. After this subdivision of the main task or procedure into instances, the software developer restricts the programming effort exclusively to one instance. When complete, this particular instance will be "cloned" by HYDRA and distributed over a user-defined number of workstations.

*Author for correspondence.

† Some tests demonstrated that powerful personal computers running operating systems like, e.g. "Linux", could be added without much effort to the local area network as well.

‡ The programming environment HYDRA is available on request, exclusively for non-commercial scientific purposes, at the Laboratory for Analytical Chemistry, University of Nijmegen, The Netherlands.

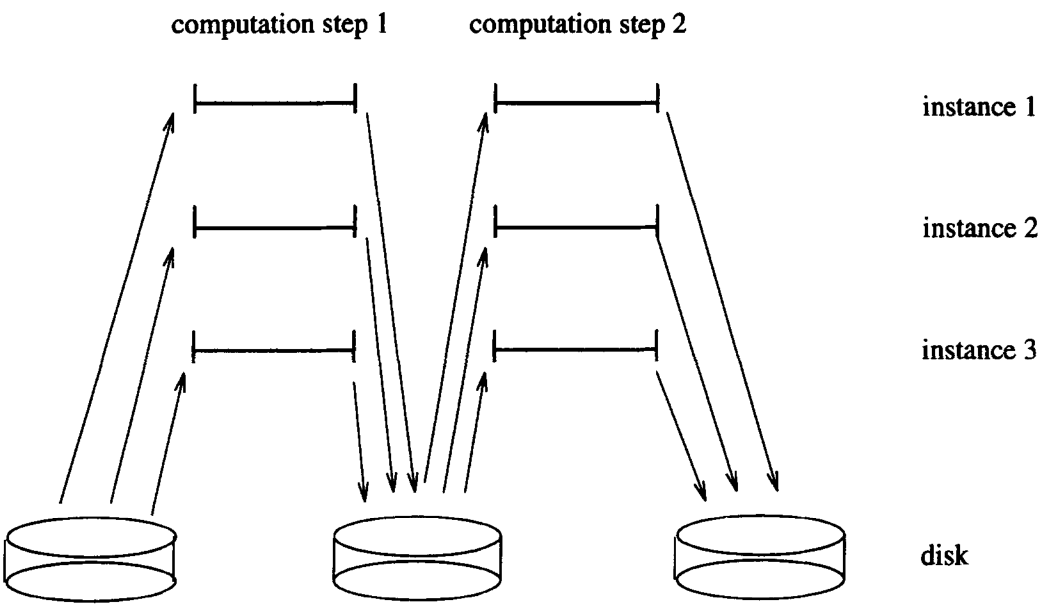

Fig. 1. Each instance reads data before starting a step and writes data afterwards.

For example, given a data base containing a huge number of multi-dimensional vectors. Assume that one has to perform a number of numerical operations on each of the vectors (e.g. calculating the inner product with some target vector). In that case, the data base can be subdivided into a number of subsets. Each subset is then treated by one program instance. Obviously, the source code for each instance is identical; the only difference is that every instance operates on a different data set. When each of the instances has treated, say, 100 vectors, the results can be stored to disk. Now, it can easily be determined by information exchange between the instances which of the subsets contained the vector most orthogonal to the target vector. This intermediate exchange of information between the instances can be done in the so-called synchronous operation mode of HYDRA (this opposed to the asynchronous operation mode, in which each instance runs independently from the others without any information exchange).

During runtime, HYDRA* takes control of each activated instance, monitors the network of workstations and will move applications to faster machines when their performance has decreased by a certain amount, traces and acts on system reboots, suspends the execution of instances if not enough workstations with a sufficient low load are available, and performs, if desired, automatically a suspension of all instances during, e.g. office hours.

Two applications of the HYDRA programming environment will be discussed in Part II [parallel cross-validation of artificial neural networks (Derks *et al.*, 1995)] and Part III [using parallel genetic algorithms for the conformational analysis of nucleic acids (Beckers *et al.*, 1995)] of this series.

2. BACKGROUND AND CONCEPT

In local area networks, the execution of software packages running for prolonged periods of time might cause severe practical problems. For instance, these generally computational intensive programs monopolize a single processor† up to a high degree, leading consequently to excessive load levels. Extremely, the situation might arise that a multi-user and multi-tasking workstation becomes hardly accessible for any other user or program. Apart from this, the continuity of the running program is not ensured due to accidental system reboots or other, usually network related, interfering phenomena. In order to avoid costly loss of results, it might be worthwhile to subdivide the total runtime into smaller steps.

This subdivision allows the control of the system status and performance between the individual steps, a procedure which is referred to as "checkpointing". If an error has occurred, the program will be restarted from the beginning of the step during which the error was encountered. Hence, only the computational effort of just one step is lost. Moreover, the principle of checkpointing guarantees a quite robust execution of an application program. However, all computations are still performed on a single processor. Since most of the modern software packages can be subdivided easily into identical subprograms (instances), there is no reason to limit the execution of a program to a single processor. Then, a number of computers

* HYDRA is named after a huge fire breathing dragon in Greek mythology. Hydra is claimed to have nine heads, one of which is mortal. When one of the other heads is chopped off, it is replaced by a new one immediately . . .

† In this paper, processor, host, machine, and workstation all refer to the same entity: a single-processor computer.

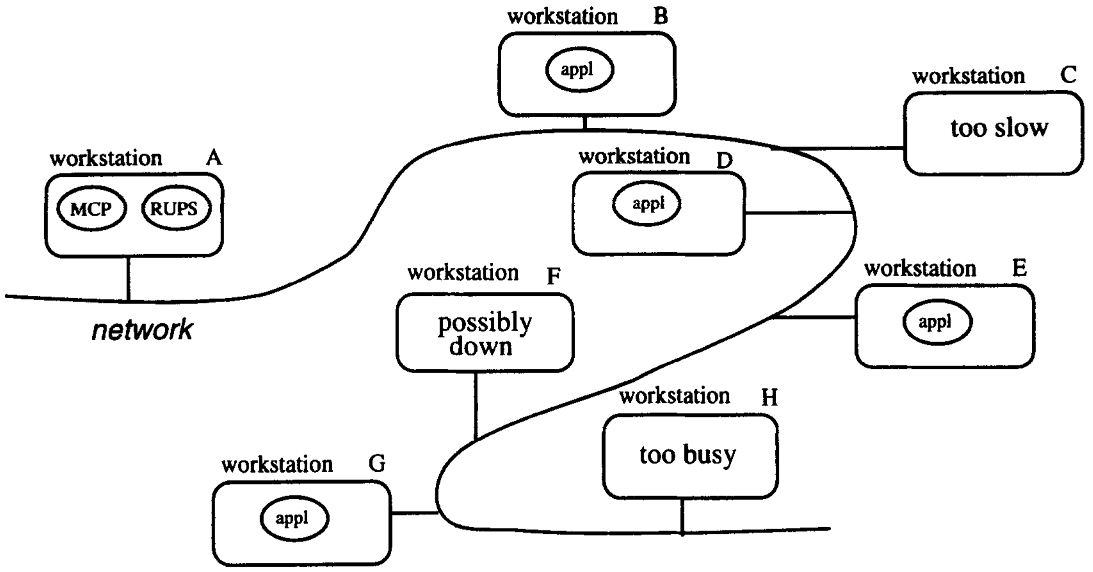

Fig. 2. Assignment of MCP/RUPS and the application instances to different workstations in a network.

of comparable performance, identical to the number of program instances, can be assigned to perform the computations in parallel, inevitably leading to a substantial reduction of the total time needed to execute the same task sequentially.

One way of establishing this type of coarse grained parallelism is described in the sequel. For each processor, at the beginning of each step, the results of the prior computing step are read from disk and loaded into the workstation memory. After completion of such a step, the new results of the computation are stored on disk thereby overwriting the formerly obtained results (see Fig. 1).

Obviously, as is the case with sequential processing, an extended implementation of checkpointing is required to ensure a robust and error-free performance of the whole network of computers as such. This, of course, introduces some overhead, especially when after each computation step information needs to be exchanged between all running instances of the application (synchronous operation mode). The software package HYDRA entails an extended implementation of the checkpointing model for both synchronous and asynchronous parallel processing in a local area network of workstations which share for communication purposes at least one file system.

Nowadays, there are usually more workstations comprised in a regular medium-sized network than the number of instances required to execute in an optimal way arbitrary application programs in parallel. HYDRA's extended checkpointing model takes this into account: it monitors the whole network of computers and selects, if appropriate, after each computing step another ensemble of workstations thereby achieving that all instances will be executed as fast as possible in each computation step. The performance criteria which are used for selecting the host machines will be discussed in detail in Section 4.6.

It should be remarked that not each application fits into the parallel scheme supported by HYDRA. Close interaction, e.g. when information exchange is required between all co-operating processors after, say, the completion of each inner loop of a single short-duration iteration, has to be avoided. In that particular circumstance, the overall performance will deteriorate, mainly due to the intensive network traffic induced by the transfer of data between the processors. Consequently, HYDRA is not suited for the implementation of *fine grained* parallel models.

3. HYDRA'S BUILDING BLOCKS

Before discussing the features and functions of HYDRA, the architecture of the HYDRA package needs a closer examination. Basically, HYDRA is built on the MCP program. MCP (Master Control Process) takes care of the assignment and control of all occupied and candidate workstations and monitors the progression of each instance. MCP is assisted by a procedure called RUPS (Remote UPtime Server), which periodically keeps track of the status of each host machine (Fig. 2). Both MCP and RUPS are executed on the same workstation, whereas the instances are distributed over a number of other hosts in the network.

Briefly, the communication between MCP and instances is established by so-called TCP/IP sockets (Transmission Control Protocol/Internet Protocol) which provide bidirectional reliable data streams (Fig. 3). Technical details are discussed in more detail in Rolf & Melssen (1993).*

The way application instances are invoked will be considered below. For the moment, it is important to

* This internal report is available on request.

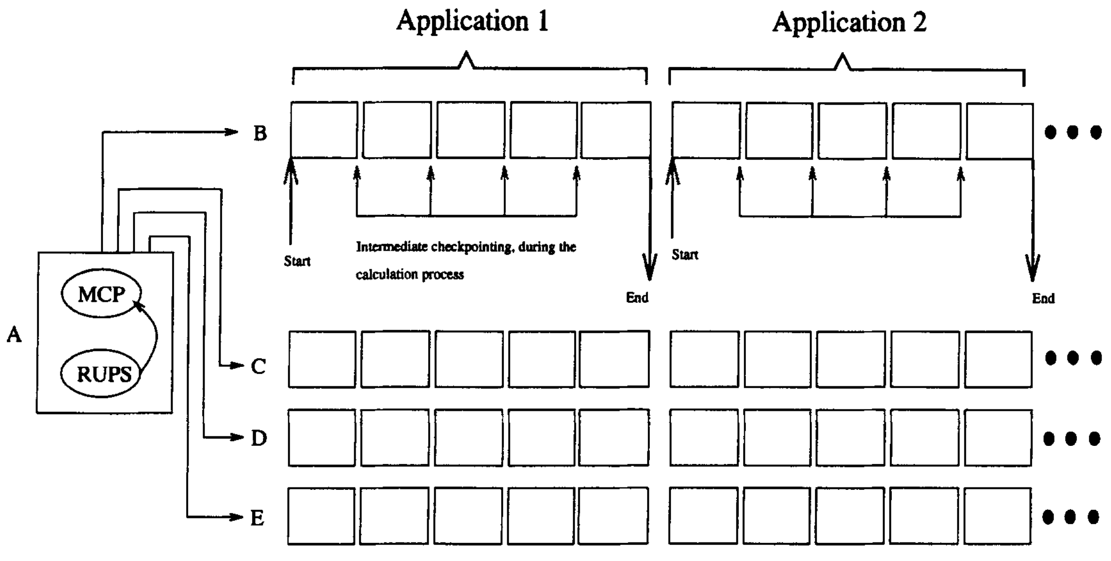

Fig. 3. Communication between MCP/RUPS (host "A") and instances (running on the hosts "B", "C", "D", and "E", respectively) via TCP/IP sockets (left section). The middle section shows the subdivision of an application into four instances (rows). Here, each instance has to perform five computation steps (columns). After the entire application has been executed, a new application can be started (right section). This illustrates how HYDRA can handle parallel as well as sequential processing in a network of workstations.

notice that a parallelized application itself does not need to do anything to initiate TCP/IP sockets for the communication with MCP.

The RUPS process polls the host machines (hosts are defined in a *host list*; this list contains active hosts as well as candidate host machines) and determines the load factor of each host. The load factor is a measure of the average number of active, running processes during a short period on one host. When a polling cycle of all hosts is done, RUPS sends a single data block to MCP which contains all the load factors. Polling of the hosts is repeated periodically by RUPS.

On receiving the Table with load factors, MCP rearranges the list of used and non-used hosts by the load factors and a hardware-independent performance criterion (see Section 4.6). The host that is expected to give the highest performance with respect to computing speed, will be assigned the first machine in the list of available hosts. Hosts that do not respond to the polling are reported by RUPS as "possibly down". In case such a host was occupied by one of the instances, the particular instance is restarted at another host. In this way, HYDRA recovers lost computation steps performed by instances which were executed on hosts that went down.

After completion of a single computation step, the instances are triggered to store the new results on disk (*write* stage) and the elapsed time needed to complete this computation step is checked for every host. Then, the optional "time-suspend function" may come into effect. If the current time is within some predefined time limits, e.g. office hours, all instances are stopped and will be re-activated when the upper time limit expires.

Additionally, a "load-suspend function" option may be invoked, to prevent instances to be executed on hosts possessing an average load factor exceeding a specified maximum value. MCP verifies the number of hosts that have exceeded this load-suspend value during the previous computation step. When insufficient hosts remain available in the host list, all instances are suspended and will be restarted when sufficient hosts will become available again. This option prevents an already busy network of computers becoming overloaded by the usually computational intensive parallel instances managed by HYDRA.

4. IMPLEMENTATION AND CONFIGURATION OF PARALLEL APPLICATIONS

4.1. Application environment

The communication of programs and error handling in Unix^{TM}-like environments is done by opening input and output streams and assigning the appropriate filepointers. These streams are the standard input (stdin), standard output (stdout) and the standard error output (stderr). In general, for most applications, the filepointers are connected to terminal-like devices [e.g. the keyboard (stdin) and a screen (stdout, stderr)]. HYDRA uses the input and output streams for transmitting and receiving protocol messages exclusively. The error message stream is redirected to a disk file.

Every instance is executed in a unique directory which has a two-digit name, e.g. "rundir/03" for the third instance. The base directory (in this example "rundir") is defined by the user. Apart from local files, each instance might share global files as well.

Table 1. The HYDRA communication protocol

Message	Symbol	Sender	Description
wait	M_WAIT	instance	instance is started and ready
read	M_READ	MCP	instance can read parameters and data
rdon	M_RDON	instance	instance has completed *read* stage
exit	M_EXIT	instance	instance is finished
calc	M_CALC	MCP	instance can start computations
cdon	M_CDON	instance	instance has finished computations
writ	M_WRIT	MCP	instance can write data to disk
wdon	M_WDON	instance	instance has completed *write* stage
trap	M_TRAP	instance	instance has found an error
stop	M_STOP	MCP	instance should stop

The application environment can be initialized by means of a user-friendly installation script.

4.2. Communication protocol

The HYDRA communication *protocol* consists of four letter messages followed by an "end-of-line" character. The latter has the advantage that, if a developer wants to test the application, the protocol messages can be entered manually as these are transmitted and received via standard terminal devices.

Table 1 summarizes the HYDRA protocol for the communication between the controlling process MCP and the instances. An instance sends the *exit* message to MCP in case it has read the parameter file(s) and detected that no further computation is necessary (for example, when a predefined number of iterations has been performed by the instance).

The *trap* message can be sent to MCP any time when an error has occurred. MCP in turn will restart the instance at another host assuming that the error will not occur again. In this way, fatal errors originating from, e.g. temporary network failures can be recovered. Of course, each *trap* message should be accompanied by a clear error statement directed to the standard error output stream; this is for control and debugging purposes.

The *stop* message can be received at any moment, but usually after the (intermediate) results have been stored on disk (in that case, a *wdon* message has been sent to MCP first). When HYDRA decides to move instances to other workstations, the instance to be moved is stopped when a single computation step is completed and, subsequently, is executed on another host.

4.3. Programming a parallel application

The programming languages C and Fortran are supported for building parallel applications with HYDRA (Fig. 4). For each programming language there is a set of header files and a library of interfacing procedures available. The header file contains amongst others the definitions of the symbolic names for the protocol messages.

When building a parallel application, the software developed only provides the procedures that perform

```
#include "mcpprotocol.h"

main()

{
    int message;

    sendm(M_WAIT);

    message = receivem();
    switch( message ) {
     case M_READ;
       ...
       break;
     case M_CALC:
       ...

    ...

}
```

```
include "mcpfprot.h"



int mes

call fsendm( M_WAIT )
call frecvm( mes )
if( mes .eq. M_READ ) then
   ...

else if( mes .eq. M_CALC) then
   ...

...
```

Fig. 4. Code fragments of parallel applications written in C and Fortran, respectively.

the specific problem-related tasks which have to be conducted by an instance in one computation step. In case a sequential version of the desired application exists, the program code for the procedures can be taken directly from the original code with just some minor changes.

4.4. Configuration of HYDRA

The behavior of HYDRA is determined by a number of parameters which are contained in a global configuration file. Most of these parameters are tunable during runtime. This has the advantage that, especially in case of long-running applications, the user has the opportunity to anticipate to varying circumstances in the computer network, for example, maintenance of a cluster of workstations or (part of) the network hardware itself. A change of a tunable parameter is detected when MCP receives the next information block from RUPS containing the average loads of all used and candidate host machines.

Table 2 summarizes the most important configuration parameters of HYDRA. A complete overview can be found in Rolf & Melssen (1993).

In order to establish a reliable value of the average load factor of each host machine during each computation step, preferably, RUPS determines the table of load factors at least four to five times during a single computation step. Because the duration of such a step is highly problem dependent, the frequency of polling by RUPS can be tuned by the RUPSINTERVAL parameter.

The last parameter, i.e. SYNCHRONOUS, determines the actual operation mode of HYDRA: synchronous versus asynchronous. In the synchronous mode, every instance has to wait after a computation step has been finished until the other instances have finished the same step too. In this way, instances are enabled to a mutual exchange of information. In the asynchronous mode, HYDRA still controls every instance, but each of these is allowed to run independently from the other ones. It should be stressed here, that in the asynchronous operation mode, HYDRA provides the user the opportunity to control even one (and thus sequential) application. In that case, HYDRA moves the application after each computing step to the best performing workstation in the network, and again, ensures a fast and reliable execution of the application.

4.5. Status monitoring

HYDRA provides a display function to monitor the HYDRA guided activities in a local area network. The status display features a list of all running instances together with their associated status and assigned host with its average load. In Fig. 5, an example of a status display window is shown. The headline of the display shows a.o. the cycle number, the overall status of MCP and the cumulative elapsed computing time of all instances. The symbols < and > indicate a decrease and increase of the load of a host machine, respectively. The number of symbols is proportional to the amount of load change. A vertical bar corresponds to a constant load. At the bottom of the display, a list of candidate hosts is given.

The display shows that some instances have completed one computation step, indicated by the status descriptor AS_CDON, whereas the other instances are still computing (status AS_CALC). Because in this example the synchronous operation mode was selected, HYDRA is waiting until each of the instances reaches the AS_CDON status. When this situation occurs, each of the instances writes the results of the computations to disk, this according to the HYDRA protocol. The next computing step is preceded by reading the most recently generated information from disk. During this stage, instances are allowed to exchange information intermediated by globally accessible disk files.

4.6. Performance measurement

For an optimal use of the computational power in a computer network, it is essential to define a hardware-independent performance criterion. Measurement of the average load factors only will not suffice

Table 2. Configuration parameters of HYDRA

Parameter	Value	Tunable change	Comments
APPLNUMBER	8	no	number of parallel instances
RUNDOMAIN	"sci.kun.nl"	no	network domain descriptor
TIMESUSPEND	yes	yes	"yes" when time-suspend is desired
TIME_TO_STOP	"08:30"	yes	time to stop instances
TIME_TO_START	"17:30"	yes	time to restart instances
LOADSUSPEND	yes	yes	"yes" when load-suspend is desired
MAXLOAD	0.2	yes	maximum sustained load of host machine
RUPSINTERVAL	30	no	number of seconds for RUPS to wait before polling the hosts again
MARGIN	10	yes	% tolerance for not acting on slower hosts and ignoring faster ones
SYNCHRONOUS	yes	no	"yes" refers to the synchronous operation mode of HYDRA

```
MCP v4 running 8 '../testappl'  cycle #4    MS_CALC  [ 32:28 ]

#  host ID    load                       status     remarks
-- ---------- -------------------------- ---------- -------
01 host_32     0.18 <                    AS_CALC
02 host_39     0.83 >>>>                 AS_CALC
03 host_01     0.59 <<<                  AS_CDON    01:10 elapsed
04 host_07     0.91 |||||                AS_CDON    01:11 elapsed
05 host_05     1.08 >>>>>>               AS_CALC
06 host_14     0.73 <<<<                 AS_CDON    01:12 elapsed
07 host_27     0.59 <<<                  AS_CALC
08 host_21     0.92 >>>>>                AS_CDON    01:19 elapsed

host_29  0.23/72        host_03  0.08/86        host_12  0.09/88
host_11  0.05/93        host_08  0.16/88        host_31  0.13/91
host_04  0.28/87        host_22  0.36/91
```

Fig. 5. The status display during computation step 4.

since certain hosts having a high load may perform considerably better than a less powerful host machine possessing a low load. These differences may originate, for example, from different internal clock rates, different sizes of internal memory, 16 or 32 bit systems, system-dependent overhead, etc.

```
12/3 11:13:23 MCP started, pid=858.
12/3 11:13:23 load suspend enabled: maxload=0.50
12/3 11:13:23 RUPS started pid=859
12/3 11:13:23 MCP tcp/ip portnr=1882
12/3 11:15:04 Execution resumed
12/3 11:15:04 start cycle 1
12/3 11:16:38 end cycle 1, 01:33 elapsed
...
12/3 11:19:03 start cycle 2
12/3 11:20:36 end cycle 2, 01:33 elapsed
12/3 11:20:56 Moved #2 from host_05 to host_01
12/3 11:21:04 Moved #7 from host_14 to host_03
12/3 11:21:04 start cycle 3
12/3 11:22:38 end cycle 3, 01:33 elapsed
12/3 11:22:39 start cycle 4
12/3 11:24:09 end cycle 4, 01:29 elapsed
12/3 11:24:32 Moved #5 from host_20 to host_21
12/3 11:24:33 start cycle 5
12/3 11:26:05 end cycle 5, 01:32 elapsed
....
12/3 11:40:59 start cycle 10
12/3 11:42:41 end cycle 10, 01:42 elapsed
12/3 11:42:48 Moved #8 from host_27 to host_29
12/3 11:42:49 MCP finished, pid=858
12/3 11:42:49 Total elapsed: 1:46:14
```

Fig. 6. Logging the behavior of HYDRA.

When a host machine has been used at least once, a so-called performance value P can be determined, which is defined by:

$$P = \begin{cases} \dfrac{T_{\text{elapsed}}}{L_{\text{aver}}} \text{ if } L_{\text{aver}} > 1 \\ T_{\text{elapsed}} \text{ otherwise} \end{cases} \qquad (1)$$

where T_{elapsed} refers to the elapsed time during one computation step, and L_{aver} represents the averaged load of a particular host obtained during a full computation step of an instance. The new ordering of the hosts in the host list is based on the expected performance, P_{exp}, of each host. For host machines that have already a peformance value (hence, such hosts have been occupied at least once by an instance), the expected performance is determined by:

$$P_{\text{exp}} = (L_{\text{act}} + 1.0)P \qquad (2)$$

where L_{act} refers to the actual load of a processor, obtained after a new polling by RUPS. Hosts that have not been occupied by an instance yet, have, of course, an undefined performance value and will be sorted by their actual load value, L_{act}. After the rearrangement is completed, the unused hosts are placed on top of the host list, in order, to reveal a reliable P_{exp} value for these hosts too. Note that here it is assumed that the average load of a processor, which is caused by the execution of one instance solely, is equal to 1.0.

4.7. Error logging

HYDRA reports all its actions, like moving an instance from one host to another. These actions are recorded in a logging file. Figure 6 depicts three typical fragments of such a logging file.

5. CONCLUSIONS

In this paper we have presented a low-cost concept of coarse grained parallel processing in a local area computer network. The main pursuit of HYDRA is to utilize optimally the available computational capacity of a network, to achieve a drastic reduction of the execution time of an application. HYDRA performs this task in such a way that every workstation remains accessible for other users of programs. Moreover, by inclusion of an advanced checkpointing strategy, a robust execution of the application is guaranteed.

Since HYDRA communicates via TCP/IP sockets, a wide variety of small- and medium-sized workstations and low-cost personal computers can be utilized to perform the execution of a parallelized application. HYDRA provides the application developer with a set of tools and libraries and supports the widely used programming languages C and Fortran. This, in addition to the simplicity of the communication protocol, facilitates the development of new applications or the parallelization of existing sequential programs. This is an advantage, as compared to porting a sequential application to, say, multiprocessor systems, because in the latter case the source code of a program usually has to be translated into hardware-specific parallel programming languages, which is very time-consuming.

The tunable behavior of HYDRA provides the user with a high degree of flexibility. The "time-suspend function" and "load-suspend function", for example, ensure that other network users are not hindered by the multiple instances of the executed parallel application.

The synchronous and asynchronous operation modes of HYDRA allow two powerful ways of parallelizing many sequential applications. This will be elucidated extensively in Parts II and III of this series (Beckers *et al.*, 1995; Derks *et al.*, 1995).

In summary, HYDRA establishes a low-cost high-performance robust easy-to-use programming environment for the parallelization and execution of computational intensive software packages.

Acknowledgements—The authors wish to thank Geert Rolf for the development of the HYDRA programming environment. E. P. P. A. Derks is supported by the Dutch Foundation for Chemical Research (SON) on behalf of the Dutch Organization for Scientific Research (NWO). The project was financially supported by the EC [project RENEGADE (ERBCHRXCT930419)].

REFERENCES

Beckers M., Derks E., Melssen W. & Buydens L. (1996) *Computers Chem.* **20,** 449.

Derks E., Beckers M., Melssen W. & Buydens L. (1996) *Computers Chem.* **20,** 439.

Dietz H. G., Cohen W. E., Muhammad T. & Mattox T. I. (1994) Compiler techniques for fine-grain execution on workstation clusters using PAPERS. 7th Annual Workshop on Languages and Parallel Computing, Cornell University.

Dietz H. G., Chung T. M., Mattox T. & Muhammad T. (1995) Purdue's adapter for parallel execution and rapid synchronization, *International Conference on Parallel Computing, 1995.*

Melssen W. J., Smits J. R. M., Rolf G. H. & Kateman G. (1993) *Chemom. Intell. Lab. Syst.* **18,** 195.

Rolf G. H. & Melssen W. J. (1993) The report on HYDRA, Internal Report. Laboratory for Analytical Chemistry, University of Nijmegen, The Netherlands.