A new control laboratory using parallel programming

S. Dormido-Canto, J. Sánchez, S. Dormido

Dept. Informática y Automática, ETS Ing. Informática, UNED
Juan del Rosal 16, 28040 Madrid. Spain

Tel. +34 91 3987151 *fax. +34 91 3988663* *E-mail*: sebas@dia.uned.es

Abstract

This paper discusses how to use parallel processing methods to solve control algorithms in real-time in the field of Control Engineering education. It is a well known fact that some types of control problems can not be dealt with in just one practical session in the lab because of their huge computational load. However, the use of low-cost clusters of workstations (COWs) and passing-message software let students program their own control algorithms and visualize the results in real-time without waiting for a long time. In this paper we describe the control of a *p*H-neutralization process using dynamic programming algorithms. The *p*H-neutralization process has been recognized as one of the most difficult single loop in process control. For this reason, this process has been used as an experiment in control teaching to show up the results obtained by using parallel techniques. Thus, this heavy-computational-load example represents a meaningful case study to demonstrate the suitableness of using parallel computing techniques to include new experiments in the control lab.

Index terms

Dynamic programming, optimal control, clusters of workstations, PVM, laboratory, *p*H-neutralization process, real-time.

1 Introduction

Since modern control theory emerged, optimization methods have been a constant in the theoretical contents of graduated courses on Control Engineering. Among all these methods, the most relevant is the dynamic programming (DP) [1] as it is a classical and powerful technique to solve several optimization problems under general conditions. Its applications are many and well-known [2], [3]: scheduling, automatic control, artificial intelligence, economics, etc.

Whereas this technique is common factor in every theoretical course on optimal control, it is a resource that has not been widely used in the control lab assignments because the calculation of the function cost is a very time-consuming task. Although dynamic programming can be applied analytically in some cases, generally the solution has to be found numerically and, now, unfortunately the problem of the dimensionality plays a key role: CPU time and memory storage requirements can become so high that, in practice, conventional dynamic programming can not be used numerically at all except to work out simple problems. For this reason, several techniques have seen developed to reduce the computational cost [2], [4], [5], [6], [7], [8], [9], [10], [11]. These techniques reduce the great disadvantage of DP, i.e. its great computational cost, but they do not solve it completely. Computational time is still very long and it does not allow us to use dynamic programming in most cases of practical interest either in industrial or educational contexts, as for example, control laboratory assignments.

One of the solutions to take advantage of dynamic programming in actual control problems, that is, to compute the control algorithm in one sampling interval, is a parallel machine. Since there is a big amount of arithmetic operations that can evaluate parallely when the dynamic programming recursive formula is calculated, the use of parallel programming techniques will allow to reduce the execution time in order to solve large-

scale dynamic programming problems. The computational theory of dynamic programming from the viewpoint of parallel computation was examined by Larson [12], but the resulting algorithms are just applicable to a very specific and expensive range of parallel computer architectures.

However, the high price of parallel computers avoids that university departments can consider seriously this solution in order to introduce dynamic programming in the control lab assignments. But, last years, decreasing prices and technological advances of personal computers have allowed to carry out parallel processing in a simple and not-so-expensive fashion by building clusters of workstations (COWs) [13], [14], [15]. Nowadays, COWs are considered a good low-cost alternative to parallel computers for many reasons (flexibility, scalability, and adaptability) but, in an educational context, the economic one stands out: the low hardware and software costs. Just hooking together a few Intel/AMD boxes by a dedicated Fast-Ethernet switch and installing any Linux distribution, and a COW will be ready to crunch numbers. Once the machine is built, the last step is to choose the most convenient paradigm of parallel programming, that in clusters it is usually by passing messages among processing nodes. There are many proprietary and public-domain message-passing systems (CMMD, Express, Fortran-M, Nx, PARMACS, etc.) but the most important and popular packages are MPI (Message-Passing Interface) [16] and PVM (Parallel Virtual Machine) [17]. Shortly, MPI is a standard specification developed by the MPI Forum, a consortium of parallel computer vendors, and PVM is a self-contained system to run parallel applications on a network of heterogeneous Linux/Windows computers.

After that, it is clear that an infrastructure for parallel processing oriented to solve optimal control problems in real-time can be afforded at low cost for many university departments. And, it will make possible to include in the traditional assignments of

laboratories new staggering and defying control projects. So far, the analysis, design, and construction of complex real-time control systems using dynamic programming algorithms in the lab were a far-fetched idea. These types of projects were forbidden as consequence of the time necessary to run the experiences when a complex process was being controlled. Now, low-cost parallel computers allow departments to widen the range of process to be controlled in the lab, being the time constrains just a matter of scalability and adaptability: to bring down the sampling interval to evaluate the control algorithm can be reached by a cluster resize.

Currently, one of the pedagogical goals of our Department is to show up to students how parallel computation may be applied to settle many types of engineering problems. For this reason, all the UNED's computer science students [18] pass a course on advanced computer architecture in which are explained the principles of parallel computing. There are many works about the contents and the scheduling of these courses [19], [20], [21], [22], where is pointed out that the engineering undergraduates have to be conversant with the tools that parallel processing offers to solve certain problems. Thus, to know parallel programming is fundamental to understand the performance that can be reached in the design and analysis of a broad range of control systems.

Looking to provide a more useful and pragmatic than theoretical view of the parallel computing, the main section of the course is focused in teaching how to construct COWs and program them using the passing messages paradigm. So, once a student passes the course, s/he must be able to build and program a COW using low cost facilities, that is, Linux as operating system and PVM as passing-messages library.

In this work is demonstrated how *p*H process modelling and the regulator design can be integrated in a cluster of PCs, bringing out a new category of control experiments to be

developed in the labs. It is further demonstrated that the experiments can be implemented in real-time.

The paper is organized as follows: A brief introduction to classical algorithms of dynamic programming is shown in Section 2. Section 3 describes the parallel implementations of these algorithms in COWs using the paradigm of passing messages. Section 4 points out shortly the main features of the cluster and software used to program the previous algorithms. In Section 5, the control via cluster of a *p*H-Neutralization process as a new control lab assignment is described. Section 6 analyses the scalability of the previous control problem and the viability of implementing it on real-time in the lab using a cluster and an improved parallel version of a classical algorithm of dynamic programming. Finally, contributions of this work are summarized.

2 Classical algorithms of Dynamic Programming

DP is based on Bellman's Principle of Optimality [1]. Basically, it states that every portion of an optimal trajectory is an optimal trajectory for a particular subproblem as it is depicted in Fig. 1.

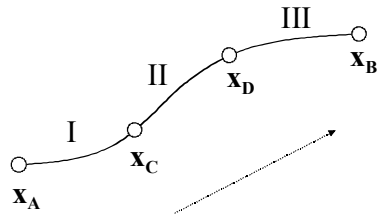

Fig. 1. Bellman's Principle Optimality: if I+II+III is the trajectory from state x_A to state x_B, according to a given cost (performance) function, then II is the optimal trajectory for the subproblem x_C-x_D.

In this case, the optimization problem can be stated as a *N*-stage decision problem defined as follows: Find the sequence of decisions $(u(1),.....,u(N))$ $(policy)$ and the corresponding sequence of states $(x(1),.....,x(N))$ $(trajectory)$ minimazing the performance or cost function,

$$\mathrm{J} = \sum_{k=1}^{N} \mathrm{L}(x(k), u(k), k) \tag{1}$$

where,

$$x(k+1) = \mathrm{g}(x(k), u(k), k) \tag{2}$$

and subject to the sets of constraints on the state and decision variables which can be formulated as,

$$x \in \mathrm{X}(k) \subset \Re^n, u \in \mathrm{U}(x(k), k) \subset \Re^m \tag{3}$$

In this problem, x is the *state variable*, X is the set of allowable states, u is the *decision variable*, U is the set of admissible decisions, k is the *stage*, and J is the cost or objective function; L represents the cost of a single stage.

If the *minimum cost function* from stage k to the end of the decision problem is defined as,

$$\mathrm{I}(x, k) = \min_{u(k),u(k+1),\ldots,u(N)} \left\{ \sum_{j=k}^{N} \mathrm{L}(x(j), u(j), j) \right\}$$

it is possible to prove using Bellman's Principle of Optimality that,

$$\mathrm{I}(x, k) = \min_{u} \{ \mathrm{L}(x(k), u(k), k) + \mathrm{I}[\mathrm{g}(x(k), u(k), k), k+1] \} \tag{4}$$

with,

$$\mathrm{I}(x, k) = \min_{u(N)} \{ \mathrm{L}(x(N), u(N), N) \}$$

for the final stage N.

In order to solve (4) numerically, the sets X and U are assumed to be finite for computational purposes; in cases where they are infinite, the set of admissible states X and the set of admissible decisions U are quantized at each stage defining a computational grid:

$$\mathrm{X}(k) = \left\{ x^1(k),\, x^2(k),\, ...,\, x^{M_\mathrm{X}(k)}(k) \right\}$$

$$\mathrm{U}(x(k),\, k) = \left\{ u^1(x(k),k),\, u^2(x(k),k),\, ...,\, u^{M_\mathrm{U}(x(k),\,k)}(x(k),k) \right\}$$

where $M_\mathrm{X}(k)$ is the number of quantized states at stage k and $M_\mathrm{U}(x(k),\, k)$ is the number of quantized decisions at stage k and state $x(k)$.

The computational method usually proceeds backwardly (*backward dynamic programming with interpolation*), as shown in Figures 3 and 4 [4]. $u^*(x^i(k),\, k)$ stands for the optimal decision at the state x^i at the stage k. The optimal decision policy is obtained for a complete family of optimization problems, i.e., for every state at all stages, and it always determines an absolute minimum within the accuracy of the computational grid as it is shown in Fig. 5.

It must be taken into account if $\mathrm{g}(x^i(k),\ \ u^j(x^i(k),k),\ \ k)$ is not a quantized state, then $\mathrm{I}(\mathrm{g}(x^i(k),\ \ u^j(x^i(k),k),\ \ k),\ \ k+1)$ has to be interpolated. It has been proven, under reasonable assumptions, that interpolation errors tend to increase almost linearly with $(N-k)$. The only way to be more accurate is the use of more quantized states and decisions, with a higher computational load.

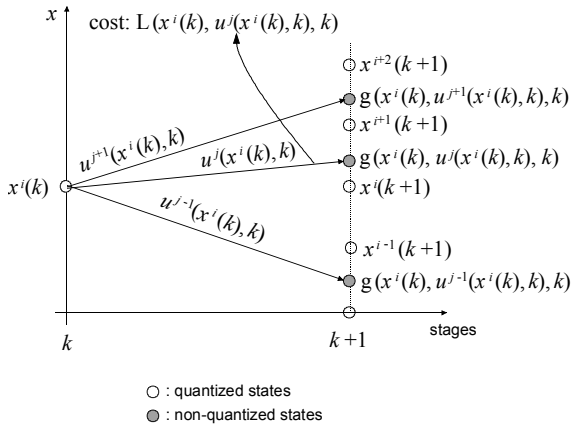

Fig. 3. Basic backward dynamic programming computational procedure at a given state $x^i(k)$.

initialize $\mathrm{I}(x,\, k) = \infty, \quad \forall x \in \mathrm{X}(k), \ \forall k$
evaluate $\mathrm{I}(x,\, N), \quad \forall x \in \mathrm{X}(N)$
for all the stages from $k = N-1$ to 1
  for all the quantized states $x^{i}(k) \in \mathrm{X}(k)$
    for all the admissible controls $u^{j}(k) \in \mathrm{U}(x^{i}(k),\, k)$
      evaluate $\mathrm{g}(x^{i}(k),\, u^{j}(k),\, k)$
      if $\mathrm{g}(x^{i}(k),\, u^{j}(k),\, k) \in \mathrm{X}(k+1)$
        interpolate $\mathrm{I}(\mathrm{g}(x^{i}(k),\, u^{j}(k),\, k),\, k+1)$
        if $\mathrm{L}(x^{i}(k),\, u^{j}(k),\, k) + \mathrm{I}(\mathrm{g}(x^{i}(k),\, u^{j}(k),\, k),\, k+1) < \mathrm{I}(x^{i}(k),\, k)$
          $\mathrm{I}(x^{i}(k),\, k) = \mathrm{L}(x^{i}(k),\, u^{j}(k),\, k) + \mathrm{I}(\mathrm{g}(x^{i}(k),\, u^{j}(k),\, k),\, k+1)$
          $u^{*}(x^{i}(k),\, k) = u^{j}(k)$
        endif;
      endif;
    endfor;
  endfor;
endfor

Fig. 4. Sequential algorithm of backward dynamic programming with interpolation.

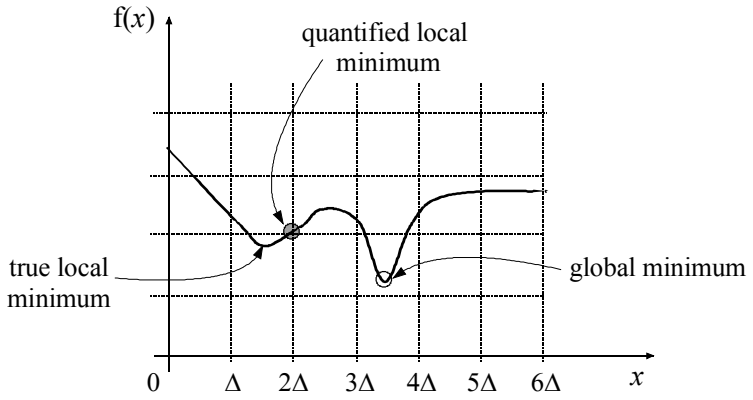

Fig. 5. A global minimum can be lost when a function is evaluated in a too coarse computational grid.

However if the inverse function g^{-1} exists,

$$\mathrm{g}\big(x(k),\, \mathrm{g}^{-1}\big[\, x(k+1),\, x(k)\,\big],\, k\big) = x(k+1)$$

an alternative sequential backward dynamic programming computational procedure without interpolation can be used (Fig. 6) [4]. As there are no errors due to interpolation, it is clear that the only way to obtain a more accurate solution is the employ of a dense computational grid.

```
initialize I(x, k) = ∞,   ∀x ∈ X(k), ∀k
evaluate I(x, N),   ∀x ∈ X(N)
for all the stages from k = N − 1 to 1
  for all the quantized states x^i(k) ∈ X(k)
    for all the quantized states x^j(k+1) ∈ X(k+1)
      u = g^{-1}(x^j(k+1), x^i(k))
      if u ∈ U(x(k), k)
        if L(x^i(k), u(k), k) + I(x^j(k+1), k+1) < I(x^i(k), k)
          I(x^i(k), k) = L(x^i(k), u(k), k) + I(x^j(k+1), k+1)
          u*(x^i(k), k) = u^j(k)
        endif;
      endif;
    endfor;
  endfor;
endfor
```

Fig.6. Sequential algorithm backward dynamic programming without interpolation.

The solution of (4) is by far the most time-consuming part of the dynamic programming computations. The approximate computation time τ, assuming there are no constraints, is:

$$\tau = \sum_{k=1}^{n} M_{\mathrm{X}}(k) \cdot M_{\mathrm{U}}(x(k), k) \cdot \Delta\tau$$

where $\Delta\tau$ is the time to solve (4) once, i.e. at one state using one decision choice. If were constraints, (4) would have to be solved less times and the actual value of τ would be smaller.

Yet any increase in both the number of states and decisions produces the fast growth of the computing time. Consequently, in order to solve many optimization problems with DP it will be necessary to resort to parallel processing. The parallel computation schemes will be discussed in the following section.

3 Parallel Dynamic Programming Algorithms

To parallelize the dynamic programming algorithms effectively, we need to know which stages are computation intensive and can be subdivided to parallelize them. Firstly, it must be noted that the evaluation of the optimal return function, equation (4), for all stages generally involves three nested iterative loops. The internal loop varies depending on algorithms with or without interpolation, as described in Figures 6 and 4. Several approaches to parallelize the dynamic programming algorithms are possible [23]. In the next paragraphs, dynamic programming parallel procedures implemented on clusters using message passing are proposed to solve optimal control problems. The master/slave paradigm has been used as programming paradigm to develop the parallel algorithms. The master is responsible for dividing the problem into small tasks, distributing these tasks among a farm of slave processors and gathering the partial results to produce the overall result. The slave processors execute a very simple code: to receive a message with data, to process the information, and to send the result to the master. The work is done in stages; each stage must finish before the work for the next stage can be generated. In this way, the master synchronizes the slaves at the end of each stage. In the following sections the classical algorithms of dynamic programming —with and without interpolation— are parallelized.

Table 1 summarizes the notations and conventions used throughout the next paragraphs.

Table 1: Notation and conventions.

Notation	Meaning
k	index of stage
m	index of processor
M	number of slave processors
N	number of stages
Δx	partition size in the space of states
Δu	partition size in the space of decisions
$(\cdot)^*$	optimal value of $(\cdot)$
$(\cdot)_i$	*i-th* component of vector $(\cdot)$
$(\cdot)^i$	*i-th* quantized value of $(\cdot)$
$\left[(\cdot)^i\right]^m$	*i-th* quantized value of $(\cdot)$ computed by the processor m
$\left[(\cdot)\right]^m$	quantized values of $(\cdot)$ computed by the processor m
$\left[(\cdot)\right]^m_{start}$	initial value in the processor m of the quantized values of $(\cdot)$
$\left[(\cdot)\right]^m_{end}$	final value in the processor m of the quantized values of $(\cdot)$

3.1 Parallel algorithms without interpolation

In sequential dynamic programming without interpolation (Fig. 6), the decision variables are not quantized. However, when the decision variables can take any value for any quantized state at the current stage, the state at the next stage is also a quantized state. For this reason, the computational grid is just defined in the set X. When this algorithm is parallelized, the parallel processing can be carried out only in the loop of the states of the stage k. The pseudocode corresponding to the master and slave processors are shown in figures 7 and 8, respectively.

MASTER

start up the parallel virtual machine: `pvm_start_pvmd( );`
start up the slave tasks: `pvm_spawn( );`
initialize $\mathrm{I}(x, k) = \infty \ \forall x \in \mathrm{X}(k) \ \forall k$
evaluate $\mathrm{I}(x, N) \ \forall x \in \mathrm{X}(N)$
send constant data to all slave processors: `pvm_mcast( );`
for $k = N - 1$ to 1
  for $m = 1$ to M
    compute $\left[x(k)\right]_{start}^{m}$, $\left[x(k)\right]_{end}^{m}$
    send to each slave processor: $\mathrm{I}(x, k), u(x, k) \ \forall x \ \forall k$, $\left[x(k)\right]_{start}^{m}$, $\left[x(k)\right]_{end}^{m}$: `pvm_send( );`
  endfor
  receive the result from each slave processor: $\mathrm{I}(\left[x\right]^{m}, k),\ u^{*}(\left[x\right]^{m}, k)$: `pvm_recv( );`
  compute and update $\mathrm{I}(x, k), u(x, k) \ \forall x \ \forall k$
endfor

Fig.7. Master computational procedure for algorithm backward dynamic programming without interpolation.

SLAVE

receive constant data from master processor: `pvm_recv( );`
receive $\mathrm{I}(x, k), u(x, k) \ \forall x \ \forall k$, $\left[x(k)\right]_{start}^{m}$, $\left[x(k)\right]_{end}^{m}$: `pvm_recv( );`
for $\left[x^{i}(k)\right]^{m} \in \mathrm{X}^{m}(k)$
  for $x^{j}(k+1) \in \mathrm{X}(k+1)$
    $u^{m}(k) = \mathrm{g}^{-1}(x^{j}(k+1),\ \left[x^{i}(k)\right]^{m})$
    if $u^{m}(k) \in \mathrm{U}(x(k),\ k)$
      if $\mathrm{L}(\left[x^{i}(k)\right]^{m},\ u^{m}(k),\ k) + \mathrm{I}(x^{j}(k+1),\ k+1) < \mathrm{I}(\left[x^{i}(k)\right]^{m},\ k)$
        $\mathrm{I}(\left[x^{i}(k)\right]^{m},\ k) = \mathrm{L}(\left[x^{i}(k)\right]^{m},\ u^{m}(k),\ k) + \mathrm{I}(x^{j}(k+1),\ k+1)$
        $u^{*}(\left[x^{i}(k)\right]^{m},\ k) = u^{m}(k)$
      endif;
    endif;
  endfor;
endfor;
send to master processor: $\mathrm{I}(\left[x(k)\right]^{m}, k),\ u^{*}(\left[x(k)\right]^{m}, k)$: `pvm_send( );`

Fig.8. Slave computational procedure for algorithm backward dynamic programming without interpolation.

3.2 Parallel algorithms with interpolation

In sequential dynamic programming with interpolation it is necessary to define a quantized computational grid in the sets X and U (Fig. 4). The parallel processing can

be carried out either in the loop of the states of the stage k, or in the loop of the decisions of the stage k. In both cases, it is necessary to use an interpolation procedure to compute the equation (4) has to be used. Both parallel codes can be found in [24].

The parallel processing algorithm of the states makes use of parallel processing carried out in the loop of the states at stage k. Each slave processor initially receives from the master a subset of quantized states at stage k. Every single permitted quantized decision has to be checked for every quantized state. Yet, in the parallel processing of the decisions the optimization procedure is carried out in two parts at each stage k. In the first part, each slave processor receives from the master only a subset of the admissible decisions and subsequently performs the optimization over all the quantized states at stage k using this subset of decisions. Thus each slave processor obtains a local optimum that is sent to the master. In the second part of the algorithm the master, once all the local optima have been gathered, computes the actual global optimum.

4 Cluster and Software Description

The cluster used in this study is composed of 16 AMD K7 processors (nodes) running at 500MHz, each one with 384MB of RAM and 7GB disk. The nodes are connected to a Fast Ethernet network through a 100Mb/s switch, making up a COW with 1 master and 15 slave processors. The operating system installed is Linux (Red-Hat 6.1). This COW is isolated from any external network, and is exclusively dedicated to solving the optimization problem.

To afford out this work, a parallel processing toolbox developed in Matlab has been used [25]: PVMTB (Parallel Virtual Machine ToolBox), based on the standard PVM. With PVMTB, users of scientific computing environments, like Matlab, in a COW with a message passing system, like PVM, can now take advantage of the rapid prototyping

nature of the environment and the clustered computing power in order to prototype High Performance Computing (HPC) applications. The user maintains all the interactive, debugging and graphics capabilities, and can now reduce execution time by taking advantage of the available processors. The interactive capability can be regarded as a powerful didactical and debugging tool.

Figure 9 shows a diagram of PVMTB. The Toolbox makes use of the PVM low-level routines and the Matlab-API (Application Program Interface) functions allow the exchange of messages among Matlab processes.

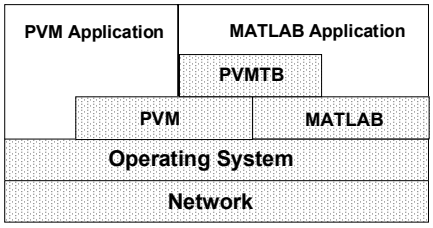

Fig. 9. Overview of PVMTB.

5 A case-study: The control of a pH-Neutralization process

The *p*H process is of great importance in the chemical industry and in waste water treatment, and it is difficult to control for a number of reasons: 1) The process is highly nonlinear; 2) It is very sensitive to disturbances near the point of neutrality; 3) It is difficult to formulate and identify a mathematical model of the process due to small amounts of polluting elements, e.g. carbonate or phosphate, change the dynamic of the process.

5.1 The experimental process

The experimental process consists, as shows Figure 10, in the neutralization of a strong acid (HCl) with strong base (NaOH) in a continuous stirred tank reactor (*cstr*) of volume (V). The acid flow (q), whose concentration is c_A (mol/l), is adjusted manually

and the base flow (u), whose concentration is c_B (mol/l), is controlled by a low flow pneumatic valve, which is regulated with a predictive controller implemented by a cluster of PCs. This feedback signal is used to provide a flow control loop at the cluster output so that this output could be regarded as the adjusting base flow, rather than the valve position. The *p*H level is measured in the outlet stream of the tank and sampled by the cluster.

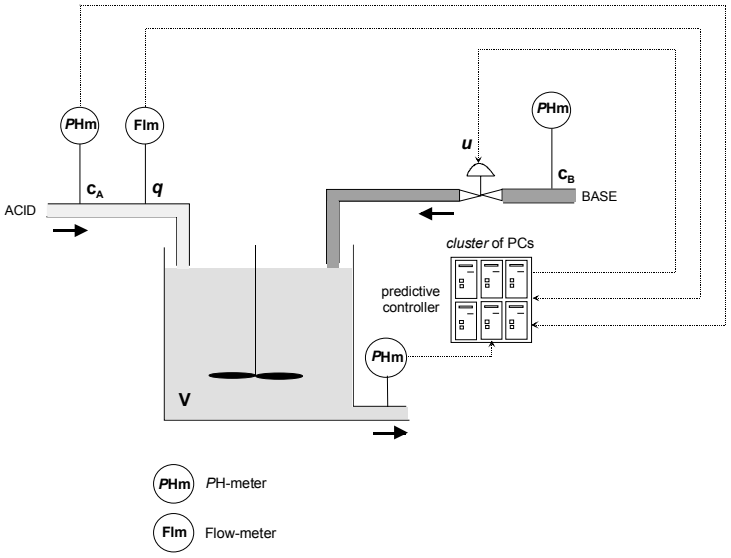

Fig. 10. *p*H-Neutralization of strong acid (concentration c_A, flow q) with strong base (concentration c_B, flow u). Tank volume is V.

Let x_A and x_B be the concentrations of acid and base in the tank respectively. The system dynamic is then given by,

$$\begin{cases} \dfrac{dx_A}{dt} = \dfrac{q}{V}(c_A - x_A) \\ \dfrac{dx_B}{dt} = \dfrac{u}{V}c_B - \dfrac{q}{V}x_B \end{cases} \tag{5}$$

and the *p*H is given by,

$$p\mathrm{H}(x) = -\log\left(\sqrt{\frac{x^2}{4} + K_w} - \frac{x}{2}\right) \tag{6}$$

where $x = x_A - x_B$ and $K_w = 10^{-14}$ (mol/l)^2 at 25ºC.

The experimental operating conditions used in our case study are listed in Table 2.

Table 2: Experimental operating conditions.

Acid flow, q	0.5 l min^{-1}
Base flow, u	0 – 0.1 l min^{-1}
Acid normality, c_A	$\approx 10^{-4}$ mol l^{-1}
Base normality, c_B	$\approx 0.5 \cdot 10^{-3}$ mol l^{-1}
Tank volume	10 l

5.2 The control system

The control purpose is to maintain the *p*H in a set point of the outlet stream by manipulating the flow of base which gets to the tank at a rate determined by the position of a valve. Thus, the position of this valve is the control input that determines the neutralization into the tank, requiring continuous adjustment under feedback control in order to achieve satisfactory results. In this case-study, the aim of the cluster of PCs is to replace a conventional PID controller.

To get students working in parallel programming for solving control problems in real-time, we have developed a predictive controller based in dynamic programming using a cluster of PCs. The control parameters in our case-study are $N_1 = 1, N_2 = 10, N_u = 1$ and $\lambda = 0$,(more details about predictive controllers can be found in [26]). Since predictive controllers make use of a process model to obtain the control signal by minimizing a given cost function, the controller is associated to an optimization problem with constraints, and it can thus be formulated as a dynamic programming problem which is described in (1), (2) and (3). So, considering (5), (6) and sampling with Euler aproximation where Δt is equal to T (sampling period), it is possible to substitute (2) in terms of *p*H:

$$p\mathrm{H}(k+1) = \frac{T \log e}{10^{p\mathrm{H}(k)-14} + 10^{-p\mathrm{H}(k)}} \cdot \left[-\frac{q}{V} \cdot \left(10^{p\mathrm{H}(k)-14} - 10^{-p\mathrm{H}(k)} \right) + \frac{c_\mathrm{B}}{V} u(k) - \frac{c_\mathrm{A}}{V} q \right] + p\mathrm{H}(k)$$

6 Experimental results

To solve the problem in real-time, students develop in the lab a parallel improved version of a parallel algorithm backward dynamic programming without interpolation known as *systematic reduction of computational grid without interpolation* [24]. In this new approach, students have to introduce a new external loop: *the number of reductions*. Consequently, they have to solve the dynamic programming problem as many times as number of reductions. Once a solution is obtained for a reduction, a band of width $2\Delta b_i$ (i goes from 1 to the number of reductions) is calculated around it. Then a new computational grid with a lower Δx_i is computed for the next reduction. In this way, a better solution with a computational complexity much lower is got it. Figure 11 depicts the procedure.

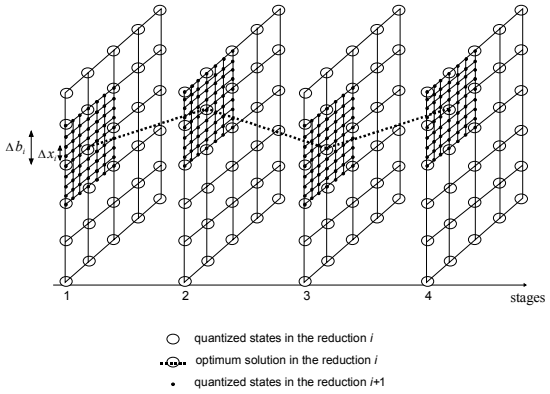

Fig. 11. Systematic reduction of computational grid without interpolation procedure with dimension 2.

Numerous simulations are afforded in the lab using different sizes to define the initial quantized computational grids in the set X. Table 3 shows the partition size (Δx) and the width of the band (Δb) in the space of states when three reductions are considered.

Table 3: Δx and Δb for three reductions.

reductions	Δx	Δb
Initial partition	0.5	8
1st reduction	0.25	1
2nd reduction	0.125	0.5
3rd reduction	0.0625	0.25
Initial partition	0.1	8
1st reduction	0.05	0.2
2nd reduction	0.025	0.1
3rd reduction	0.0125	0.05
Initial partition	0.01	8
1st reduction	0.005	0.02
2nd reduction	0.0025	0.01
3rd reduction	0.00125	0.005

Figure 12 shows some results for differents set points in the *p*H control with $\Delta x = 0.5$ and $\Delta x = 1$ as initial partitions and an initial value of *p*H equal to 4.

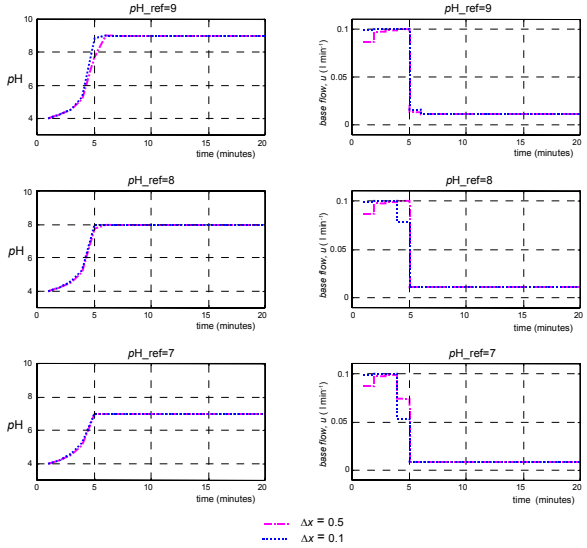

Fig. 12. Results of the simulations.

Table 4 shows the average times for each sampling time with differents sizes of the initial partition in a single processor system.

Table 4: Average time (in seconds) for each sample time in a single processor system.

t (Δx = 0.5)	t (Δx = 0.1)	t (Δx = 0.01)
0.75	10	944

A measure of the relative performance of a multicomputer system is the speedup factor, $S(M) = t_s / t_p$, where t_s is the execution time using one processor and t_p is the execution time using a computer with M processors. But, also, in a message-passing system, the time to send messages must be included in the total execution time of a problem. Thus, the parallel execution time (t_p) is obtained by adding two elements: the computation time (t_{comp}), and the communication time (t_{comm}): $t_p = t_{comp} + t_{comm}$.

As the COW is dedicated to the resolution of the optimization problem and isolated from any external network, the standard deviation of t_p is very small and can be ignored. Only mean times will be considered.

Figure 13 shows the average time (in seconds) for each sample time and the speedup obtained as the number of processors is increased. In accordance with the obtained results, the following general observations can be pointed out: 1) for coarse initial partitions in the space of states (Δx = 0.5), the speedup with less than five slave processors is very low. In Figure 13(a) can be observed how the *Amdahl's limit* (maximun number of processors for solving a problem in the minimum time) is reached for M = 4. Therefore, the result with M = 10 is worse than the time obtained by a single processor. 2) for fine initial partitions in the space of states (Δx = 0.1), the computational burden has been increased and the speedup reaches a saturation point from M = 13 as it is depicted in Figure 13(b). 3) finally, for very fine initial partitions in the space of states (Δx = 0.01), the speedup is quite better, almost linear (Figure 13(c)). In this case, since the computational burden has been increased considerably, the

computation part (t_{comp}) predominates over the communication part (t_{comm}) in the parallel execution time (t_p) as it can be appreciated in Figure 14.

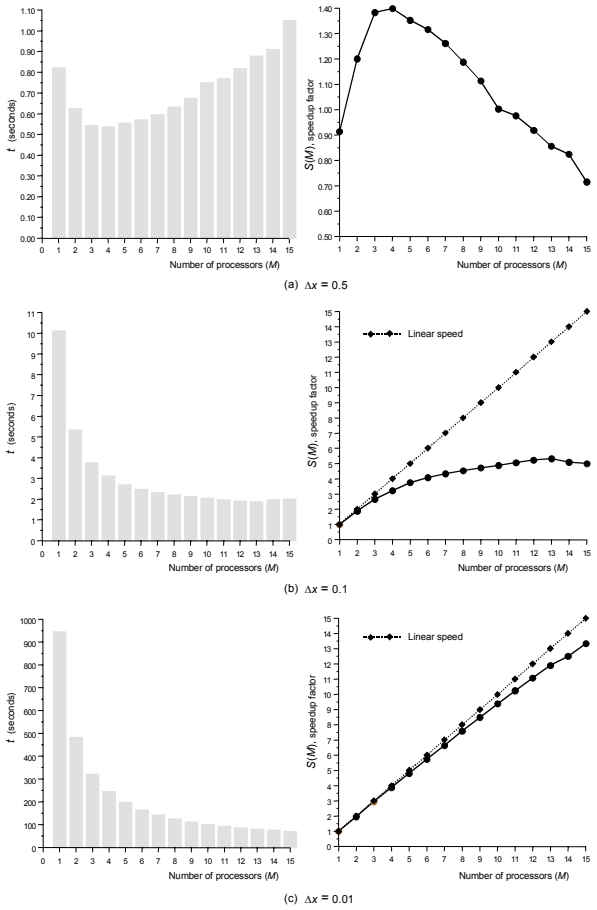

Fig. 13. Average times (in seconds) in each sample time and speedup factor.

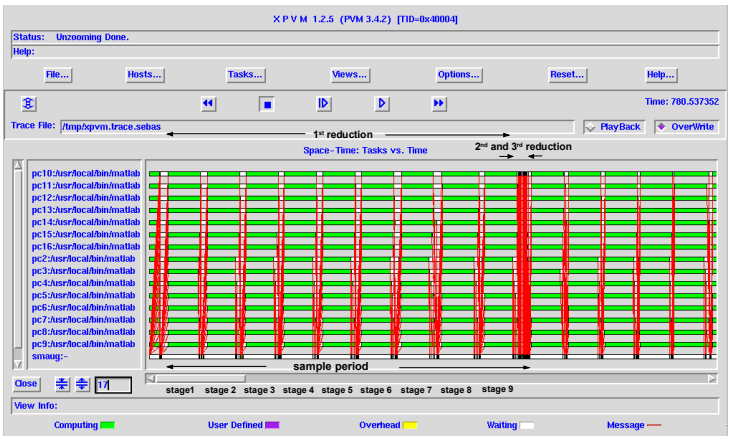

Fig. 14. Chronogram of *p*H control with a size initial partition Δx = 0.01.

One of the more important points to guarantee the viability of implementing the *p*H control in real-time is the response time of the controller. It is clear this time must be

lower than the sampling time. In this case-study, the controller response time is 60 seconds [24] and the sampling time in a single processor is shown in Table 4. For $\Delta x = 0.5$ and $\Delta x = 0.1$ as sizes of the initial partition, the control of the *p*H could be solved in real-time with just a single processor. However, with $\Delta x = 0.01$ is necessary to use a cluster of PCs. In the considered experiment, with $M = 15$ the sampling time is 70 seconds (Figure 13(c)), this means that by adding two or three processors to the cluster is possible to control the *p*H-Neutralization in real-time.

7 Conclusions

This paper has shown how feasible is to use methods of parallel processing to solve control algorithms in real-time in the field of Control Engineering education. Decreasing prices and technological advances of personal computers have allowed to carry out parallel processing in a simple and not-so-expensive fashion by building clusters of workstations (COWs). Now, low-cost parallel computers allow university departments to widen the range of process to be controlled in the lab. So, it makes possible to include in the traditional assignments of laboratories new staggering and defying control projects.

In the experimental work presented here it is demonstrated how *p*H process modelling and regulator design can be integrated in a cluster of PCs for a new control experiment in the labs. It is further demonstrated that the experiments can be implemented in real-time.

Acknowledgements

This work has been supported by the Spanish CICYT under grant DPI2004-01804.

References

[1] R. E. Bellman, "Dynamic Programming", *Princeton University Press*, New Jersey, 1957.

[2] R. E. Bellman and S. E. Dreyfus, "Applied Dynamic Programming", *Princeton University Press*, New Jersey, 1962.

[3] A. Grama, A. Gupta, G. Karypis and V. Kumar, "Introduction to Parallel Computing, Design and Analysis of Algorithms", second edition, *Addison Wesley*, 2003.

[4] A. P. de Madrid, S. Dormido and F. Morilla. "Reduction of the Dimensionality of Dynamic Programming: A Case Study", *American Control Conference - ACC99*. San Diego, USA, 1999.

[5] R. E. Larson and A. J. Korsak, "A Dynamic Programming Succesive Approximations Technique with Convergence Proofs", Part I, *Automatica*, vol. 6, pp. 245-252, 1970.

[6] A. J. Korsak and R. E. Larson, "A Dynamic Programming Succesive Approximations Technique with Convergence Proofs", Part II, *Automatica*, vol. 6, pp. 253-260, 1970.

[7] L. Cooper and M. W. Cooper, "Introduction to Dynamic Programming", *Pergamon Press*, 1981.

[8] R. E. Larson and J. L. Casti, "Principles of Dynamic Programming. Part II: Advanced Theory and Applications", *Marcel Dekker*, *Inc.*, New York, 1982.

[9] L. Moreno, L. Acosta and J. L. Sánchez, "Design of Algorithms for Spacial-time Reduction Complexity of Dynamic Programming", *IEE Proc.-D*, vol. 2, pp. 172-180, 1992.

[10] M. Sniedovich, "Dynamic Programming", *Marcel Dekker*, *Inc.*, New York, 1992.

[11] A. P. de Madrid, S. Dormido, F. Morilla and L. Grau, "Dynamic Programming Predictive Control". *IFAC*, *13th Triennial World Congress*, 2c-02, pp. 279-284, San Francisco, USA, 1996.

[12] R. E. Larson and E. Tse, "Parallel Processing Algorithms for the Optimal Control of Nonlinear Dynamic Systems", *IEEE Transactions on Computers*, C-22, vol. 8, pp. 777-786, 1973.

[13] G. F. Pfister, "In Search of Clusters", *Prentice Hall*, New Jersey, 1998.

[14] R. Buyya, "High Performance Cluster Computing", *Prentice Hall*, vol. 1 (Architectures and Systems), New Jersey, 1999.

[15] R. Buyya, "High Performance Cluster Computing", *Prentice Hall*, vol. 2 (Programmings and Applications), New Jersey, 1999.

[16] M. Snir and W. Gropp. "MPI: The Complete Reference", *The MIT Press*, Cambridge, Massachussetts, 2001.

[17] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancher and V. Sunderam, "PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing", *The MIT Press*, Cambridge, Massachussetts, 1994.

[18] Homepage, UNED [Online]. Available: http://www.uned.es/webuned/home.htm

[19] T. Hintz, "Introducing Undergraduates to Parallel Processing", *IEEE Trans. Educ.*, vol. 36, pp. 210-213, Feb. 1993.

[20] F. C. Berry, "An Undergraduate Parallel Processing Laboratory", *IEEE Trans. Educ.*, vol. 38, pp. 306-311, Nov. 1995.

[21] J. A. Youssefi and K. Zemoudeh, "A Course in Parallel Processing", *IEEE Trans. Educ.*, vol. 40, pp. 36-40, Feb. 1997.

[22] B. Wilkinson and M. Allen, "A State-Wide Senior Parallel Programming Course", *IEEE Trans. Educ.*, vol. 42, pp. 167-173, Aug. 1999.

[23] R. E. Larson and J. L. Casti, "Principles of Dynamic Programming. Part I: Basic Analytic and Computational Methods", *Marcel Dekker*, *Inc.*, New York, 1978.

[24] S. D. Canto, "Programación Dinámica Paralela: Aplicación a Problemas de Control", *Ph.D. Thesis*, Dpto. Informática y Automática (UNED), Madrid, 2002, (in Spanish).

[25] J. Fernández, A. Cañas, A. F. Díaz, J. González, J. Ortega and A. Prieto "Performance of Message-Passing Matlab Toolboxes", *Springer-Verlag*, vol. 2565, pp. 228-241, Heidelberg, 2003.

[26] J. M. Maciejowski, "Predictive Control with Constraints", *Prentice Hall*, 2001.

S. Dormido-Canto received his MS degree in electronic engineering in 1994 from the Universidad Pontificia de Comillas University (ICAI) and his Ph.D. degree in physics from the UNED (Universidad Nacional de Educación a Distancia) in 2001. He joined at UNED Department of Computer Sciences and Automatic Control as an Assistant Professor in 1994. His current research and teaching activities are related with the analysis and design of control systems via intranet or internet, high performance interconnection networks for cluster of workstations and optimal control.

J. Sanchez received his Computer Sciences degree in 1994, from Madrid Polytechnic University and his Ph.D. in Sciences from UNED (Universidad Nacional de Educación a Distancia) in 2001. Since 1993, he has been working at UNED Department of Computer Sciences and Automatic Control as an Assistant Professor. His current research interests are the design of new systems for control education, virtual labs, telepresence, multimedia, and the use of the Internet in education.

S. Dormido received his Physics degree from Madrid Complutense University (1968) and his Ph.D. from Country Vasc University (1971). In 1981, he was appointed Full Professor of Control Engineering at UNED. He has supervised 25 PhD Thesis and co-authoring more than 150 conference papers and 100 journal papers. Since 2002 is President of the Spanish Association of Automatic Control, CEA-IFAC. His scientific activity includes various topics from the control engineering field: computer control of industrial processes, model-based predictive control, robust control, modeling and simulation of hybrid systems and control education with special emphasis on remotes and virtual labs.