7. Parallel Image Processing

7.1 Presentation

This example is an illustration of the possibilities of distributing complex processes between different agents to substantially reduce the overall execution time of the application, especially in image processing and multimedia applications. The example in Chapter 3 shows how to implement a communication between several agents, the processing of which depends on the results of neighbouring agents. In addition, to this *horizontal communication* between slave agents, there still exists *vertical communication* between the master agent and the slave agents according to the master–slave model supported by PVM.

7.1.1 Function of the Master Agent

In this application, the role of the master agent can be summarized as follows:

1. cut the image to be generated into blocks of almost identical size;
2. create a matrix of $N \times N$ slave agents;
3. transmit the dimensions of the block of image to be generated to each slave agent;
4. collect the different blocks of slave agents and display them in an Xwindow.

The fundamental idea is to divide the Xwindow generated by the master agent into a matrix of $N \times N$ blocks. Each slave agent will therefore have the task of generating the contents of the corresponding block and transmitting it to the master for display. There is a bijection between the window zones and the agent slaves (Fig. 7.1). The master agent plays the role of a client process compared to the server X of the host system.

This application displays two types of interactions:

- **client–server**: the master demands the services of the server X to manage the display window for the image to be generated;
- **master–slave**: the master creates a set of agent slaves. Each slave will be responsible for the management and generation of part of the image (Fig. 7.1). The slaves are then destroyed.
7.1.2 Functions of Slave Agents

Each slave agent has the single role of generating the contents of the part of the window that is associated to it, following logical cutting. It must, in fact, generate the colour at each point (pixel) of its block. Calculation of the colour at each point depends on the nature of the image processing application. Where we are concerned, the processing of each agent is limited to calculating the colour of each point by using a simple calculation function which will be explained later. In this way, the processing generally carried out by a slave can be symbolized by the following simple algorithm:

```
Agent_i
  Somuchas Active Do
    Generate dialogue between the neighbouring agents to calculate \( \Psi(t) \);
    For each point \( x_j \) of my block \( b_i \) do
      \( x_j(t) = \text{colour}(\Psi(t), t) \);
    endfor
    transmit \( B_i(t) \) to the master;
  Endsomuchas
Endagent;
```

The "colour" function calculates the value (colour) of the pixel \( x_j \) at instant \( t \) depending on the \( \Psi \) context vector. This vector takes into account the initial conditions at the borders of the block at instant \( t \) (this relates to the example of heat propagation along a wire in Chapter 2 (Section 2.3.5) to be able to calculate the colour of the internal points of the block at instant \( t \). The vector of the \( \Psi \) context generally requires communication with neighbouring agents to update itself before calculating the new values of the internal points. The colour function represents the progression or development law of any phenomenon: direction and displacement speed of an object in an animation image, simulating the progression of a natural phenomenon (clouds, desertification of certain regions, illness etc.). We have introduced the notion of time into the "colour" function because the image needs refreshing over time. The time interval is chosen depending on the nature of the application. These points will not be discussed any further as they fall outside the scope of this book.