Parallel systems development in education: a guided method

E. Luque, J. Sorribes, R. Suppi, E. Cesar, J. L. Falguera and M. Serrano
Departament d'Informàtica
University Autonoma of Barcelona
08193 Bellaterra (Barcelona)
Spain
iinfd@cc.uab.es

Abstract

Our objective is to show a parallel system development method within the educational area based on the Parallel System Development Environment (PSEE). To do so, first we define the development cycle proposed by PSEE, and the theoretical model underlying it. Secondly, we use an example (matrix multiplication algorithm) to illustrate all the method steps, which are: Algorithm Choice-Criteria, Problem Presentation (how to present the problem to the student), Modelling the algorithm (using the theoretical model), Programming (using an appropriate parallel language), Functional Simulation of the program, Behavioural Simulation (algorithm and architecture), and Trace Visualization and Evaluation. This paper gives a description of the parallel systems development method, based on our own experience with PSEE (Parallel System Evaluation Environment) [1].

1 Introduction

The main goal is to show the facilities provided by the PSEE in parallel algorithms development, within the educational area.

PSEE is an user interactive graphical environment, developed to provide a tool for studying the behavior of distributed memory parallel systems. This environment allows parallel program development and also simulation and evaluation of parallel algorithms on parallel architectures. PSEE is able to manage the main system parameters so as to achieve the highest fitting grade between the architecture and the algorithm. This involves a cyclical development process among programming, simulation, visualization, taking measures and modifying parameters.

To accomplish each stage of this cycle, several tools have been implemented.

The Programming stage is performed using the WinPie Tool [4]; this tool accepts applications written in a high level parallel language, and translate them to sequential code. This code is a composition of the original application code and a set of functions used to perfom the functional simulation of the application.

Moreover the tool is also able to obtain a graphical representation of the algorithm, that could be used in the behavioural simulation.

The Simulation stage is accomplished by creating or modifying the synthetical graphs of algorithms and architectures (Graph Editor), by allocating algorithm tasks to processors (Mapping Tool), and finally by doing the behavioural simulation (Simulation Tool) [2].

The Visualization stage is achieved by generating a set of measures on each simulation trace in order to measure the intrinsic parallelism, the dynamic parallelism as well as the scheduling, routing and clustering policies quality (Graph Tool) [5].

From this analysis, the user may decide to change any of the established policies, or may try to simulate onto another different hardware architecture, or may even discover that any portion of the parallel algorithm generates problems, or take into consideration any other aspects from the analysis. Then the user may introduce any change and follow the development cycle again

The model underlying these tools is called the Weighted Behavioural Graph[1] (WBG for short), and is defined as a directed graph, where nodes represent sequential code segments, and edges represent dependency relations (data or control) between those nodes. This model allows both the simulation of the algorithm behavior and its allocation to different processing elements. It also allows the estimation of its performance using different execution, scheduling and routing policies.

Let's give an example that illustrates the development method described above.

2 An example of the development method

The method goes from the algorithm choice up to the results analysis, through problem presentation and modelling, and through the stages described in previous section.

2.1 Algorithm choice criteria

Like sorting and searching, matrix multiplication is a fundamental component of many numerical and non-numerical algorithms. On the other hand, Matrix multiplication provides all sorts of opportunities for parallelization on multiprocessors.

The user should learn how different architectures work with a single algorithm in order to decide the best one. The character of the algorithm depends heavily on the architecture on which it is to be implemented. Therefore, we have selected a Cube-Connected model (the best suited).

For example to analyze communication costs, simple Cube-connection allows routing policies and mapping algorithms to be easily implemented.

2.2 Problem presentation: matrix by matrix multiplication

The presented problem is based on the matrix multiplication algorithm for the hypercube hardware topology given in [3], restricted to 2 x 2 matrix and three dimensional hypercube, which is represented by a 2x2x2 matrix of processors (to fit the algorithm requirements).

This algorithm has the following functionality (figure 1):

1. The elements of the matrices involved in the operation (call it A and B) are distributed over the n^3 processors.
2. The products C(i, j, k) = A(i, j, k) * B(i, j, k) are computed.
3. The sums $\sum_i$ C(i, j, k) are computed.

Note: Indexes i, j and k are referred to the processors matrix

Figure 1

This description is presented in a simple way in order to ease the algorithm comprehension and to fit it to the WBG model.

2.3 Modelling

Once the specific algorithm has been understood by the student, it should be posed, based on the WBG model, to adapt the algorithm to the set of tools that will be used to its study (see figure 2).

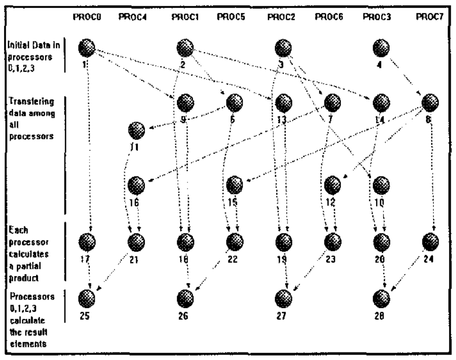
PROC0
PROC4
PROC1
PROC5
PROC2
PROC6
PROC3
PROC7
Initial Data in processors 0,1,2,3
Transfering data among all processors
Each processor calculates a partial product
Processors 0,1,2,3 calculate the result elements
1
2
3
4
9
6
13
7
14
8
11
16
15
12
10
17
21
18
22
19
23
20
24
25
26
27
28

Figure 2

The algorithm under study is highly oriented to a particular architecture, and this architecture is well modelled by our hardware model (message passing architectures), so, a good way to plan the WBG based solution is to give a main role to this architecture.

The idea is to draw a graph that follows the WBG model rules, defining first the processes needed to send the data from any processor to its neighbours, then the processes needed to receive this data, and finally the processes that do the calculations.

2.4 Program

Once the student has been able to build a graph based on the WBG model, then he knows which task is performed by each process (graph node). The next step is straight-forward, and consists of codifying these tasks using the Pascal language and a set of parallel primitives on WBG model basis.

These primitives define a programming model based on WBG model, and they are:

- Subroutine (Input Policy, Output Policy, parameters....). which allows the process definition.
- Token (Channel, Data). which allows information transfer.
- Init_Token(Channel, Data). to fire the program execution.
- Active. variable that keeps for each node, how many times it has been activated.
- Varhold. which allows the definition of some variables that do not lose their values between node activations.

As an example of code that implements one of the graph nodes see (figure 3):

```
Subroutine no17 (and;and; n17a11, n17b11:integer);
        var C0 : integer;
      begin
        C0 := n17a11 * n17b11;
        token ( n25C0, C0, 'n' );
      end;
```

Figure 3

2.5 Functional simulation

The program written by the student is functionally tested using the WinPie Tool. WinPie offers a programming environment that generates an execution file. This result should be used to perform a sequential execution of the parallel algorithm into a single-processor machine, simulating its parallel functional behavior. The functional simulation allows the student:

- to check the correct functionality of the parallel algorithm.
- to get the values of the main parameters from a parallel algorithm: Node execution time and communication volume. As an example, for the subroutine showed above (in figure 3) we obtain the information shown in figure 4.

2.6 Behavioural simulation

Once the algorithm and hardware have been characterized, and a mapping between them has been defined, the behavioural simulation, which is event driven, can be done.

The first step may consist in performing the algorithm simulation without the architecture; this is equivalent to assuming that we have an architecture with the same structure of the algorithm, where the communications are no time consuming, and where only the nodes computation volumes are traced because no

Process Id : *NO17*
Class : Normal,
Processing Speed : 1
Priority : 0
Input Policy : AND
Execution Policy : AND
Inputs: Arc from Node : 2, Arc from Node : 8
Outputs: Arc to Node : 25, Communication Cost : 1,
Probability : 0

Figure 4

hardware influence exist. In this case we assume the ideal architecture for the algorithm.

Performing this simulation the student may see all the real parallelism expressed in his/her algorithm, because this algorithm is not affected by the hardware.

The second step may consist of simulating several times with different hardware characterizations, and using different scheduling, mapping and routing policies, in order to compare them.

In each simulation, a trace file with the measured events may be generated, in order to analyze the simulation results.

2.7 Visualization tools

The *Data Filters and Visualization Tool* allows the evaluation of the simulator output traces, by generating a set of measures on each trace by means of a predefined set of filters.

The visualization and evaluation tool design is based on a simple and practical data dealing concept, in order to provide an easy management of a great information volume (generated on the Trace File by the Behavioural Simulator), and a sensitive graphical data visualization.

The ultimate target of this presentation of the information is to help the user to study the correctness of the parallel algorithm, the performance of the given architecture and the efficiency of the scheduling, clustering and routing policies on them.

3 Conclusions

With the description made in this work, we have described how the user, using PSEE, may follow the whole development cycle of designing and programming a parallel algorithm. Simulating it oo different architectures, setting several scheduling, clustering or routing policies. Analyzing the correctness and performance of parallel systems, going back to any point of the cycle to modify any parameter. So the user may learn with a practical experience in an interactive way.

References

1 PSEE for Windows User's Guide. Computer Science Dep. University Autonoma of Barcelona (UAB). 1989-1995.

2 Cesar, Eduardo. Entorno de Simulación de Sistemas Paralelos sobre una Interface Gráfica Estandar (GUI). MsC Project UAB. 1995.

3 G. Akl, Selim. *The Design and Analysis of Parallel Algorithms.* Prentice-Hall International Editions. 1989.

4 Falguera, José Luis. Entorno Interactivo para la Simulación Funcional de Programas Paralelos. MsC Project UAB. 1995.

5 Serrano, Maria. Herramienta de Visualización y Evaluación de la Traza de Simulación. MsC Project UAB. 1995.