Applying Data Mining Techniques to Tune System Scheduling*

Antonio G. Lomeña, Marisa López Vallejo and Antonio García Quintas
Departamento de Ingeniería Electrónica
Universidad Politécnica de Madrid
Madrid (Spain)
e-mail: {lomena,marisa,gquintas}@die.upm.es

Abstract

This paper presents the application of data mining techniques to find out what parameters have a bigger impact on the scheduling of heterogeneous systems. In particular, static scheduling is implemented by means of a list-based scheduling algorithm. The influence of the different parameters that configure the scheduling platform has been analyzed by means of the Kohonen Self Organizing Maps. This technique has shown to be a powerful tool for the analysis and visualization of high-dimensionality data. Additionally, we have proved the usefulness of data mining methodologies to study in depth other design automation problems characterized by many data-influenced attributes to tune.

1 Introduction

Hardware-Software co-design methodologies have achieved widespread acceptance over the past years. The need to reduce design flow cycles while satisfying the ever increasing throughput and area constraints of the applications, has boosted research into design automation techniques. More specifically, Hw-Sw co-design appeared as a mean to fulfill these requirements altogether by exploiting the synergism between the hardware and software development cycles. In these complex heterogeneous systems a key implementation goal is to meet a given performance constraint. Consequently, system scheduling plays a crucial role in this arena.

Moreover, scheduling participates in many phases of the co-design methodology. First, during the partition stage a scheduler must be used to supply feedback information about the timing of a given partition [1]. Since partition algorithms usually employ iterative refining techniques, the utilization of fast, efficient schedulers becomes a matter of prime importance. Later on in the design flow, schedulers are again used to schedule both the software instructions and the hardware operations of the partitioned system. Hence, the knowledge of the main factors that affect the performance of the different scheduling algorithms becomes of crucial importance to improve the applicability of the co-design methodology. This is the reason why recently some efforts have been devoted to the improvement of particular aspects like the optimization of bus access [2] or the incorporation of speculative execution [3].

However, the high number of parameters involved, the subtleness and complexity of their relations as well as the large amount of empirical data that must be dealt with to achieve a good statistical confidence, have precluded traditional analysis techniques from achieving a more thorough knowledge of the problem.

Recently, data mining techniques have emerged as a suitable tool to tackle large databases to discover the hidden relations that connect the multiple parameters of a given process [4]. These techniques become an ideal candidate to explore the relations, symmetries and connections of the different variables that appear in the scheduling problem.

This paper aims at discovering the main factors that determine the scheduling features of a given set of tasks executed on a heterogeneous system (with standard processors and dedicated co-processors). By identifying what parameters produce a major impact on the scheduling performance, appropriate actions could be taken in order to improve the scheduling cost. Also, this knowledge can be applied to build estimation functions to predict the scheduling result of a given problem. This can be used to guide the search in heuristic scheduling algorithms.

To carry out the scheduling data exploration we have employed a set of Kohonen self-organizing maps (SOM) [5] as analysis tool. This technique has shown to be very well suited to handle large amounts of multi-parameter information allowing to extract the similarities, regularities and correlations underlying in different problems [6]. Further, once these maps have been trained, they can be used as estimation functions for the aforementioned purposes.

SOMs have been widely employed in the monitoring and modeling of complex industry processes like fault diagnosis [7]. Also, they have been useful in several engineering applications such as pattern recognition, text and image analysis, financial data analysis or adaptive resource allocation [8]. Finally they have successfully been employed in areas such as knowledge discovery [9] and data mining [4].

The paper has been structured as follows. Section 2 describes the scheduling problem. In section 3 the main aspects of the SOM methodology are introduced. Section 4 deals with the problem of adapting the scheduling data to be processed by a SOM. Section 5 explains the decisions adopted to select the different configuration parameters of the self-organizing

* This work is funded by CICYT project TIC2000-0583-C02-02

algorithm. Experimental results are analyzed in section 6 and, finally, some conclusions are drawn.

2 Task scheduling model

Task scheduling consists in optimally allocating a series of activities to a finite set of resources under the existence of a group of constraints [10]. The final goal of the scheduling is to minimize the execution latency. The present work is oriented to DSP applications, that are characterized by tasks of deterministic duration. Since sporadic arrivals are not considered, the schedule can be calculated statically, meaning that all the scheduling decisions have been computed before the application execution. In our environment, every task can only be allocated to a processor, and every processor can only execute a task at once.

Input to the scheduling platform is a directed and acyclic graph where vertices stand for the tasks (basic computation units) and edges represent data and control dependencies. Theses dependencies mean that a given task t_i may only start after its predecessor task, t_j has already finished its execution and has delivered the results to t_i.

The scheduler takes into account the timing estimates of every vertex in the graph and the dependencies among them. Every task is labeled with several attributes, being of interest from the scheduling point of view the following: the hardware execution time (ht_i) and software execution time (st_i). Edges have also associated a communication value ($t_{comm}(i,j)$). As output, the scheduler gives the design latency, T_p, and the communication cost produced in the Hw-Sw interface. In this sense, an added complexity appears when communication times are taken into account. Two dependent tasks executed in different processors will employ a communication time that will be added to the final scheduling latency. When executed in a same processor, communication costs between two tasks are negligible.

The *target architecture* consists of several standard processors running the software and several hardware co-processors (based on ASIC or FPGA), all characterized by a different processing speed. Tasks will have assigned a given implementation, either software or hardware. A software task can only be executed in a software processor and the behavior is analogous for hardware tasks.

Scheduling is implemented by means of a list-based scheduling algorithm [10]. Given a set of ready tasks (those whose predecessors have already been scheduled) this technique assigns a priority to each of them and takes the task of highest priority as the next to schedule. The selection criteria that have been employed in our experiments are:

Task priority criteria: three priority schemes can be chosen to select the next task to schedule:

1. Greatest number of children: gives a higher priority to that ready task which has more children.
2. Greatest number of descendants: gives a higher priority to that ready task with a greater number of descendants, counting each descendant just once (no matter that the descendant can be reached several times by different paths).
3. Greatest number of weighted descendants: this scheme is alike the previous criterion but it counts each node as many times as it is accessed through different paths.

Allocation criteria: two criteria have been implemented to decide which processor to choose for the execution of a given task[1]:

1. Earliest started first: allocates the task to that processor which is ready to compute first.
2. Earliest finished first: allocates the task to the processor where it would finish first (taking into account the communication times).

So far we have described the main features that characterize the scheduling problem. As it has been previously said, the high number of parameters involved as well as the large amount of empirical (sometimes ambiguous) data obtained during the different scheduling experiments, make difficult to extract well defined rules about the global scheduling behavior. We will circumvent this hindrance employing data mining techniques as a method to explore the hidden information in the available data. Next section delves into the foundations of one of these techniques.

3 Self organizing maps

Kohonen self organizing maps (SOMs) are a kind of unsupervised neural networks widely employed for classification and categorization purposes. This becomes specially useful when dealing with an unknown number of classes, since Kohonen maps allow to determine the most suitable amount of categories the input data could be fitted into [11].

Each neuron i of the SOM is represented as an n-dimensional weight vector, where n is the dimension of the input vectors (attributes that characterize the problem under study). The number of neurons of the map determines the granularity of the transformation to carry out what, in turn, affects the accuracy and generalization power of the net.

During the iterative training process, the SOM approximates the probability density function of the input patterns [5] by means of its neuron weights. To achieve a good convergence, it is important to dispose of a large quantity of input patterns. Otherwise, it might be necessary to repeat the training several times using the same patterns.

The mining of scheduling data carried out in the present work has been developed following the methodology proposed by Kohonen [12] which divides the process into four different phases:

[1] The first allocation criterion is the classic ASAP scheduling but the reader should not confuse the second criterion with ALAP scheduling

Net configuration: this phase defines the dimension and topology of the map as well as the neighborhood function (step or Gaussian function) that will be used during the training.

Training: after randomly setting up the neuron weights, the input patterns are used to train the map, repeating the self-organizing algorithm a number of cycles for each pattern (this number depends on the total amount of available patterns). At every iteration, each neuron computes the Euclidean distance from the input pattern to its weight vector. The winning neuron is that of least distance. All the neurons will update their weight vector according to the equation

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)]$$

where t is the discrete time (an integer modeling the current iteration), m_i is the weight vector corresponding to the neuron i, x is the input pattern and h_{ci} is a neighborhood function around the winning neuron. Two neighborhood functions have been explored in the present work:

Step function

$$h_{ci}(r_i, t) = \begin{cases} \alpha(t) & if \ ||r_g - r_i|| \leq R_o \\ 0 & if \ ||r_g - r_i|| > R_o \end{cases}$$

where r_g is the location of the winning neuron, r_i is the location of the current neuron and R_o is a constant radius. $\alpha(t)$ is the learning rate, a monotonically decreasing function whose exact form is not critical.

Gaussian function

$$h_{ci}(r_i, t) = \alpha(t) exp\left(-\frac{||r_c - r_i||^2}{2\sigma^2(t)}\right)$$

where $\alpha(t)$, r_i and r_g have the same meaning as in the step function. $\sigma(t)$ is a time decreasing function and defines the neighborhood radius. The learning rate function is either a linearly decreasing function or an inverse function of the form $\alpha(t) = A/(B+t)$.

In summary, the main parameters to configure in the training phase are:

- iteration number (*rlen* parameter).
- initial radius of the neighborhood function (R_o o $\sigma(t_o)$).
- learning rate function and its initial value $\alpha(t_0)$.

Quantization error computation: It provides an indication of the accuracy achieved by the trained net when it acts as an estimation function. This error is computed introducing again the same input vectors used during the training and calculating the distance from the winning neuron to the pattern. The quantization error is then the average of all these errors.

Visualization: Finally, the interpretation and analysis of results is performed by means of unified distance matrices (u-matrices) [13]. This method consists in creating a matrix of Euclidean distances among the weight vectors of adjacent neurons.

4 Application of SOMs to the scheduling problem

4.1 Attribute definition

One of the most important phases of the data mining methodology consists in defining the scheduling data attributes that will be studied. In our case, we can distinguish between:

Architectural attributes, which characterize the system to be scheduled. We will consider: number of available hardware and software processors, task priority criterion, processor allocation criterion, number of tasks in the graph to be scheduled, tasks graph connectivity (maximum number of children of a given task), task graph depth (maximum number of tasks composing a path in the graph), mean and standard task execution times and, finally, mean and standard inter-task communication times.

Performance attributes, which measure the quality of the obtained schedule. We have considered: schedule latency (total delay of the scheduling), communication time (total communication time spent by the scheduled tasks[2]), communication number, penalty time (defined as that part of the communication time that produces an increase of the scheduling latency[3]), penalty number, and CPU processing time. Table 1 shows the actual values used for each architectural attribute. The combination of all of them gives a total of 5832 different scheduling problems. After scheduling all these configurations, 5832 different patterns were obtained, each one of them composed of 17 components (coding architectural and performance attribute values). This set constitutes the raw database of scheduling information which will be employed to train the SOM. A second set, known as test set, was also generated to employ it for validation purposes as will be explained in section 5.1.

4.2 Input data pre-processing

To correctly train the SOM it is necessary to preprocess the raw input data calculated in the previous section to adequate them to the net and training properties. In general it is convenient [6] that input data exhibit:

1. a good distribution (Gaussian or uniform input data distribution).
2. similar data range for all the net inputs.
3. data values bounded inside the interval allowed by the activation function of the neurons.

To assure the second property we generated two sets of data from the raw input patterns. Specifically, a global row scaling was applied (the transformation being individually applied over sets composed of the

[2] Remember that the communication time of tasks allocated to the same processor is neglected

[3] This is a subtle concept, due to the fact that a given inter-task communication may not increase the scheduling latency when it has been established while other operations, performed by other tasks, are actually imposing a minimal bound to the latency

Table 1: Input pattern component values

Number of Hw processors	{1; 4}
Number of Sw processors	{1; 4}
Priority criteria	See section 2
Allocation criteria	See section 2
Task number	$\{n < 50; 50 \geq n \leq 100; 100 \geq n\}$
Connectivity values	{1; 2; 3}
Task execution times. Gaussian Distribution	
Mean task execution	50
Standard deviation values	{5; 25; 250}
Task communication times. Gaussian Distribution	
Mean communic. values	{5;50;500}
Communication standard deviation factors	0.1; 0.5; 5.0
Execution and Communication times Uniform Distribution	
Mean task execution	50
Mean communic. values	{5; 50; 500}

same component of every input pattern). We explored two kinds of transformations:

standarization: the component variables are transformed to provide zero mean and variance unity.

normalization: a vector normalization is carried out over the standardized data so that every component will be scaled in the interval $[-1, +1]$.

These transformations produce patterns where every component has the same importance, since their variability and range of values have been levelled.

It was found out that the SOMs trained with standardized data produced the best results. When using raw data, the disparity of the samples caused frequent map divergence. On the other hand, the normalized data tends to originate well grouped but not very homogeneous maps. Therefore the patterns ultimately used for the final experiments were only standardized.

5 Configuration of SOM parameters

As mentioned before, three main aspects have to be tackled to configure the SOM previous to the training phase: net configuration and neighborhood and learning rate function selection. The following sections explain the details about these issues.

5.1 Net configuration

We determined the ideal net size performing several experiments to calculate the quantization error for each generated map. The larger the number of net neurons, the greater the accuracy will be. However, a net size excessively large as well as a training performed for too many cycles can lead to over-training the net. In this case the map will be too accurately adapted to the introduced examples, decreasing the generalization capacity of the net.

Consequently, the quantization error alone is not a valid indicator to settle the net size. Thus, we generated a second group of examples and used them as a test set to restrain the net from specialization. The methodology we have followed is:

Table 2: Quantization Errors

NET SIZE	QE	TE
10×20	2,345520	2,343124
40×20	1,975246	1,976300
60×40	1,714744	1,725268
100×75	1,397963	1,417198

1. to calculate the quantization error of the trained net using the test set as input patterns. This quantization error will be called test error (TE).
2. to increase the size of the map and train it as long as the test error keeps decreasing.

The experiments were performed in two stages with a step neighborhood function and a linear training rate of value $\alpha(t_0) = 0.5$ and $\alpha(t_0) = 0.002$ respectively. The first stage allows a global organization of the neurons while the second stage is a refining phase of local specialization. The nets were trained for 50000 cycles at each stage. During the first stage the neighborhood radius was established close to the largest neural map dimension, and between 8 to 10 neurons during the second stage. The final results appear in table 2. It can be seen that the test error monotonically decreases without showing any variation in this trend. This indicates that the over-training situation was not reached.

According to the quantization error the most suitable net would be the 100×75 one, since it presents the highest precision without producing over-training. Nevertheless, some studies on supervised neural networks [14] show that in order to reach a generalization error $\epsilon = 0.1$ (10%) the learning pattern number must be around $p = 10w$, where p is the pattern number and w the net neuron weight number. Additionally, we must take into account the effect known as *curse of dimensionality* [15] by which the number of data necessary to achieve a good training exponentially grows with the input space dimension.

As mentioned before, 5832 patterns were available with 17 components each one. Using a 100×75 neuron net $100 \times 75 \times 17 \times 10 = 1275000$ training vectors would be needed to achieve a 10% generalization error [14]. This number results almost unreachable while the 40×20 only requires 136000, which is more affordable. Yet, since we only dispose of 5832 patterns it will be necessary to iterate them a high number of times, which was controlled by the *rlen* parameter.

Finally, it might be argued that the curse of dimensionality could be circumvented by increasing the iteration number. However, this technique might lead to over-training since the net could become finally specialized to those patterns. Furthermore, the training could require long computing times. For instance, the 100×75 net has 10 times more neurons than the 40×20 net. Further, if we would additionally increment the iteration number in one magnitude order we would get 100 times greater computing times, which could become prohibitive (the training of a net of 40×20 neurons took about 30 minutes in

Table 3: Learning rate and neighborhood functions

Learning rate function	Neighborhood function
linear	step
linear	Gaussian
inverse	step
inverse	Gaussian

a 200 Mhz Pentium with 32 MB).

Taking into account all these considerations, the configuration finally chosen for the ensuing experiments was a 40×20 neuron net. This represents a good trade-off between quantization error, generalization error and training time.

5.2 Neighborhood and learning rate function selection

To ascertain which was the most suitable combination of the neighborhood and learning rate functions, all the possible arrangements were explored. They appear in table 3. Matrix-u analysis was employed as a criterion to decide on the resulting combinations. It was found that the samples formed better defined clusters in the *Gaussian neighborhood* maps than in the *step neighborhood* ones. However, the data samples are more homogeneous in the clusters belonging to the maps obtained with the *step* function. The homogeneity criterion seems more useful than the clustering level, since it allows to decide the meaning of each group, according to the type of samples that predominate in the cluster. Therefore, the step neighborhood was chosen to carry out future experiments.

Similarly, the maps produced with the combination step-linear appeared more homogeneous than in the case of step-inverse functions. Consequently the step-linear combination was finally chosen for the final tests.

6 Analysis of experimental results

The group of experiments was composed of two sets consisting of:

- patterns corresponding to tasks with Gaussian execution and communication time distributions.
- patterns of tasks with uniform execution and communication times.

The analysis of the patterns exhibiting uniform time distributions produced conclusions similar to the results obtained for Gaussian distributions. This implies that the type of distribution does not have a major impact on the scheduling performance. Hence the next section will only focus on the analysis of patterns with Gaussian distribution.

6.1 Component matrices

The behavior of each component can be represented in a neuron matrix (*component matrix*) where clear zones stand for neurons specialized on high values for that attribute, while dark colors indicate a specialization in low values. For instance, figure 1 represents the number of tasks of the scheduled graphs and figure 2 the corresponding communication number. The relations among different attributes are manifested by the zones of their maps that overlap. It can be seen how the zones of clear neurons overlap over the two maps. This means that there exists a correlation between the high values of these two attributes. Obviously, the scheduling of graphs with a high number of tasks produces results exhibiting a large number of communications.

Next, we will explain the main correlations found among the different component matrices. We are mainly interested in discovering the relations that bind the architectural attributes with the performance attributes.

- The graph depth level is the factor that influences most the number of communications and penalties. As the depth level grows, both the number of communications and of penalties rise. Also, these two variables seem to be tightly bound since the increment of communications makes the penalties grow.
- The schedule latency, communication and penalty times are closely related. This means that parameters increasing one of these attributes will also increment the others (hereafter we will refer to these times as *delay times*). However, the relation of these times with respect to the number of communications and penalties is slighter. Therefore, there are many cases in which a high number of communications does not waste long times (does not rise the latency, communication or penalty times).
- All the architectural attributes affect the delay times of the schedule. However, the most determinant parameters seem to be the graph task number as well as the mean and standard deviation of the inter-task communication times. High values of these attributes will commonly override other architectural factors. The standard deviation times for the task executions did not show a very high incidence over the delay times.
- The greatest values of the delay times corresponded to schedulings performed with:
  - the *earliest start first* allocation criterion.
  - the *greatest number of children* or the *greatest number of successors* priority criteria.

  This makes *earliest finish first* and *greatest weighted successor number* the most efficient criteria.
- The zone of greatest delay times does not overlap the areas of maximum connectivity and, moreover, it is coincident with regions were the maximum number of hardware and software processors was used. This behavior can be explained because when the number of processors is increased, an excessive distribution of the tasks among the processors is produced (thus incrementing the scheduling cost).

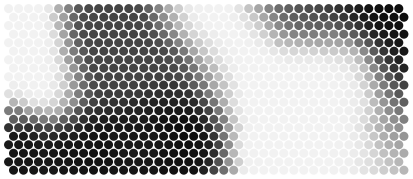

Figure 1: Task number

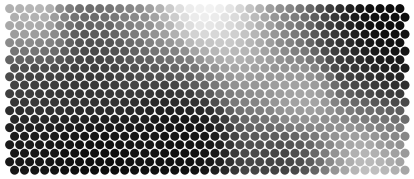

Figure 2: Communication number

- The behavior of the delay times with respect to the connectivity can be explained noticing that the cost is maximum when the connectivity value is at a medium level. If the connectivity increases, the presence of a larger number of processors makes the scheduling more fluent (what implies a decrease in the latency).
- Two zones were found where graphs with high number of tasks (and great depth level) together with high connectivity values did not produce a high latency. These regions where scheduled with the lowest number of software processors and the highest number of hardware processors. This architecture thus becomes very appropriate to tackle large sized, strongly connected scheduling graphs. Consequently, the communication overhead produced when a high number of software processors is employed, seems to be more determinant for hardware processors (the speed increment provided by an increase in the number of hardware processors counteracts the communication overhead).
- Finally, the highest values of CPU time were closely related to graphs exhibiting high depth levels. The list scheduling algorithms are therefore more affected by this factor than by others such as the connectivity.

7 Conclusions

The present work has applied the SOMs methodology to perform data mining of scheduling data for heterogeneous systems. The knowledge acquired will have evident applications for a better understanding of the complex relations and factors involved in the scheduling process. Also, the trained nets can be employed as scheduling estimators in heuristic scheduling algorithms.

The most important conclusion that has been drawn is the little influence of the probabilistic distribution (Gaussian or uniform) of the task execution and inter-task communication times on the scheduling results. On the contrary, factors such as the actual inter-task communication times (rather than the task execution times), the processor architecture of the system or the priority and allocation criteria were found the most determinant. Other interesting correlations described in the paper were also applied to the scheduling platform to tune the scheduling results.

References

[1] A. Kalavade and E. A. Lee. The Extended Partitioning Problem: Hardware/Software Mapping, Scheduling and Implementation-bin Selection. *Journal of Design Automation of Embedded Systems*, 2(2), March 1997.

[2] P. Eles, A. Doboli, P. Pop, and Z. Peng. Scheduling with bus access optimization for distributed embedded systems. *IEEE Transactions on VLSI Systems*, 8(5):472–491, April 2000.

[3] G. Lakshminarayana, A. Raghunathan, and N. K. Jha. Incorporating speculative execution into scheduling of control-flow intensive designs. *IEEE Trans. on Computer-Aided Design*, 19, March 2000.

[4] J. Vesanto. *Using SOM in Data Mining*. PhD thesis, Helsinki Univ. of Technology, Finland, April 2000.

[5] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9), september 1990.

[6] B. Martín del Brío and A. Sanz Molina. *Redes neuronales y sistemas borrosos*. Ra-Ma, Textos Universitarios, 1997 (in Spanish).

[7] E. Alhoniemi, J. Hollmn, O. Simula, and J. Vesanto. Process monitoring and modeling using the self-organizing map. *Integrated Computer-Aided Engineering*, 6(1), 1999.

[8] H. Tang and O. Simula. *The optimal utilization of multi-service scp*, pages 175–188. Cahpman & Hall, 1996.

[9] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in knowledge discovery and data mining*. AAAI Press/MIT Press, 1996.

[10] H. El-Rewini, T. G. Lewis, and H. H. Ali. *Task scheduling in parallel and distributed systems*. Prentice Hall, 1994.

[11] T. Manninen and J. Pirkola. Classification of textual data with self-organizing map. *Raahe Laboratory of the University of Oulu*, 1998.

[12] T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen. SOM_PAK the self-organizing map program package. *Laboratory of Computer and Information Science, Helsinki University of Technology*, 1995.

[13] A. Ultsch and H. Siemon. Kohonen's self organizing feature maps for exploratory data analysis. *Proc. INNC'90, Dordrecht, Netherlands*, 1990.

[14] E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1, 1989.

[15] C. M. Bishop. Neural networks and their applications. *Rev. Sci. Instrum.*, 65(6), 1994.