

# Software Pipelining using a Hierarchical Social Metaheuristic

Abraham Duarte<sup>1</sup>, Felipe Fernández<sup>2</sup> and Ángel Sánchez<sup>1</sup>

<sup>1</sup>Dept. Informática, Estadística y Telemática, ESCT-URJC

Campus de Móstoles, 28933, Madrid Spain

e-mail: {a.duarte, an.sanchez}@escet.urjc.es

<sup>2</sup>Dept. Tecnología Fotónica. Facultad de Informática. UPM

Campus de Montegancedo, 28860, Madrid, Spain

e-mail: Felipe.Fernandez@es.bosch.com

**Abstract:** *Software pipelining is a compile-time scheduling technique that overlaps successive loop iterations to achieve instruction-level parallelism. This paper presents a highly efficient algorithm for determining the iteration bound of cyclic data flow graphs and their optimal loop scheduling for the unlimited resources case. To solve this problem, a new general metaheuristic has been used. This metaheuristic is inspired in some hierarchical social structures and processes. Different variants of the proposed algorithm have been implemented and successfully benchmarked.*

**Keywords:** *Metaheuristics, software pipelining, extended retiming, scaled retiming.*

## 1. Introduction

In parallel processing, synchronous Data Flow Graphs (DFG's) are frequently used to specify real time applications performing repetitive tasks, such as signal and image processing, and to analyse the available parallelism. A DFG is a directed graph, where directed edges model the precedence constraints between nodes. External data to the system are sampled periodically and the corresponding period is called iteration period. System throughput is improved by reducing the sampling period. The shortest iteration period is called iteration bound and the corresponding scheduling is termed rate-optimal. A DFG contains a critical circuit that determine the iteration bound. An active research has been carried out to achieve the maximum throughput or minimum iteration period by means of the appropriate scheduling. In review [5] various scheduling methods are discussed for the unlimited resources case. This paper is also concerned with this unlimited resources case. The corresponding process can be additionally used as the first stage of the limited resources scheduling problem.

Scaled retiming [4] is here used to model the static schedule of a DFG, to reorganise the scheduling of an iteration and to solve the corresponding software pipelining problem.

Hierarchical social (HS) algorithms [2][3] is a new metaheuristic for finding an optimal solution in a discrete domain. HS algorithms are inspired in some hierarchical social structures and processes in order to solve multilevel discrete optimisation problems. This metaheuristic is here utilised to efficiently solve the considered loop-scheduling problem.

## 2. Scaled slack graph

To analyse the software pipelining and loop scheduling, this paper uses DFG's [5]. The loops of a computation programme can be defined as an arc two-weighted digraph:

$$G = \{ V, A, [D], [T] \}$$

where  $V$  is the ordered set of nodes that represents the set of tasks ( $|V|=n$ ),  $A$  is the set of oriented edges or arcs of the corresponding graph and represents the set of precedence relations among the nodes ( $|A|=m$ ),  $[D]$  is a column  $m$ -vector of delays, associated with each arc of the graph (called algorithmic delays) and  $[T]$  is a column  $m$ -vector of computation times also associated with each arc of the graph (called computational or architectural delays).

Sha et al. [5] have introduced a rational retiming method, for the corresponding rational scheduling graph, named extended retiming. This model is introduced since the traditional retiming does not always achieve optimal schedules [5]. In this paper an equivalent integer extended retiming model is used, which is named scaled retiming [4]. The involved slack graph  $G_\alpha$  is given by:

$$G_\alpha = \{ V, A, \alpha[D]-[T] \}$$

where  $\alpha$  is the iteration period or maximum computational time for all circuits in  $G$ .

### 3. Hierarchical Social Metaheuristic for the Software Pipelining Problem

HS algorithms [2] are mainly inspired in the hierarchical social behaviour observed in a great diversity of human organisations. This metaheuristic have been successfully applied to several problems [2][3]. The key idea of HS algorithms is a simultaneous optimisation of a set of disjoint solutions obtained on a dynamical domain partition (society). Each subset of the domain partition (group of the society) contains a feasible solution and these groups are initially random distributed through the solution space. By means of the social evolution strategies: intra-group cooperation and inter-groups competition, better solutions are obtained. In this social evolution, the groups with lower quality tend to disappear. The process usually ends with only one group that contains the best solution found.

#### 3.1 Individual and group objective functions

Each individual of a group has an individual objective function  $f_1$ . Also each group of the society has a group objective function  $f_2$  that is shared by all individuals of a group. In the referred scheduling problem for a graph  $G = \{ V, A, [D], [T] \}$ , these group and individual objective functions are:

- The *group objective function*  $f_2(i)$ , for each group  $i$  (subset of the domain partition), is given by the corresponding iteration period:

$$f_2(u, i) = f_2(i) = \alpha_i = \frac{\sum_{(u,v) \in C_i} t(u, v)}{\sum_{(u,v) \in C_i} d(u, v)}$$

where  $C_i$  is the corresponding circuit of the group  $i$ .

- The *individual objective function*  $f_1(u, i)$ , for the input node  $u$  (element of a group  $i$ ) of each active arc  $(u, v)$ , is given by the scaled retiming function and computed in backward order:

$$f_1(u, i) = f_1(v, i) - f_2(i) * d(u, v) + t(u, v)$$

The HS algorithms try to optimise one of the objective functions  $f_1$  or  $f_2$ , depending on the operation phase. During the cooperation phase, each individual of a group  $i$  tries to improve its group objective function  $f_2$ . During a competition phase, the cohesion of loser groups disappeared and their individuals only try to improve their individual objective function  $f_1$ , activating relations with winner groups. This way, the graph partition is successively modified.

For the scheduling problem, the domain partition is carried out selecting (activating) only one output arc for each node of the graph, in each phase of the corresponding evolution process.

#### 3.2 High Level Pseudo-Code

A general high-level description of an HS metaheuristic is shown in Figure 1, where the main search procedures are: Cooperation Intra-group Strategy and Competition Inter-group Strategy. The first one is oriented to cooperation movements among individuals that belong to the same group. The second one is oriented to competition movements among individuals that belong to different groups.

```

Procedure Hierarchical_Social_Algorithm(S)
Var
  S=(V,A,D,T)={{v}{a}{d}{t}}:Initial_Society_graph;
   $\Pi$ ={gi}:Groups structure that establishes the domain partition;
  F1={f1}:Individuals objective function structure;
  F2={f2}:Groups objective function structure;
Begin /* Begin social evolution */
   $\Pi$ =Get_initial_random_partition(S);
  Repeat /* group evolution*/
    F2 = Compute_F2( $\Pi$ ); F1 = Compute_F1( $\Pi$ ,F2); /*Objective functions*/
    For each gi in  $\Pi$ 
      If f2(i)=max { f2(k)  $\forall$ k } Or Cooperation_phase
        Then Cooperation_Strategy(gi, F2) Else Competition_Strategy(gi, F1);
      End For
    Update_groups_structure( $\Pi$ );
  Until termination_condition_met; /*End of social evolution*/
  Return( $\Pi$ ); /* Approximate optimal solution */
End Hierarchical_Social_Algorithm

```

**Figure 1.** High-level pseudo-code for Hierarchical Social Algorithms.

In the software pipelining problem, the Cooperation Strategy (Figure 2) is oriented to improve the scaled retiming vector  $f_i$  of each individual in a group. For a given group  $g_i$ , a node maximize the distance from itself to a node which acts as reference. These movements are constrained to a group  $g_i$ .

```

Procedure Cooperation_Strategy(gi, F2)/* for the software pipelining problem*/
Var
  NodesInGroup: Individuals of group gi;
  f2(i): Group Objective Function of group gi;
  {f1(u,i)}: Individual Objective Function on each individual in group gi;
Begin
For v = 1 to NodesInGroup
  /*Select the arc a = (u,v)  $\in$  A that maximizes the scaled retiming vector*/
  (u,v) = arg max {f1(u,i): f1(u,i) = t(u,v) - f2(i)*d(u,v) + f1(v,i),(u,v) $\in$ A};
  gi = Insert((u,v),gi) /*Insert the new arc, removing the old arc*/
End For
End
End Cooperation_Strategy

```

**Figure 2.** Pseudo-code of Cooperation Strategy.

Competition Strategy (Figure 3) allows an individual to move to another group. The individuals belonging to groups with lower value of iteration period  $f_2$  can change their group in order to increase their group objective function.

```

Procedure Competition_Strategy(gi,F1)/* for the software pipelining problem*/
Var
  NodesInGroup: Individuals of group gi
  i,j,k: 1,...,NodesInGroup
Begin
For u = 1 to NodesInGroup
  /*Remove the sole arc whose origin is the node u*/
  RemoveActualArc(gi);
  /*Select the arc a = (u,v)  $\in$  A that maximizes the iteration period*/
  (u,v)= arg max {max (f2(v,k)), $\forall$ k} /*Look for its Neighborhood*/
  gi = Insert((u,v),gi)/*Insert the new arc, removing the old arc*/
End For
End
End Competition_Strategy

```

**Figure 3.** Pseudo-code of Competition Strategy.

## 4. Software Pipelining Example

This section shows an example of how to obtain a rate-optimal software-pipelining program by means of the described HS scheduling algorithm. Figure 4.a shows the initial strongly connected DFG considered. In this figure: circles represent tasks labelled from A to P; arcs represent precedence relations and black solid bars represent algorithmic delays. Notice that computational delays have been omitted in the picture for clarity. All DFG arcs have a computational delay equal to 1 unit, except the output arcs of nodes K, L, O and P that have a computational delay of 3 units. These nodes are indicated in grey in this figure. The rest of figures of this example show the consecutive steps applied.

In this figures the following notation is used: all the individual objective functions are displayed inside the nodes; the group objective functions are shown near to the circuit of each group; if there are several groups, the nodes of each group are represented with different grey levels. For each iteration and each individual, the next active relation is shown in grey.

In each step of the considered scheduling algorithm, each individual (node), following a specified strategy, activates only one relation (output arc). Figure 4.b shows a first partition obtained using a random strategy. In this case there is only one group and the obtained circuit is composed by the nodes I, M. Later, each node tries to improve its individual objective function changing the actives arcs, (Figure 4.c). Afterwards, two groups are formed and therefore the nodes in the loser group try to find the best relation (arc) with the winner group.

At this step, the society (third partition in Figure 4.d) is again integrated in one group. In the next stage of the society (fourth partition in Figure 4.e) is once more separated into two groups, so each individual searches the best relation in order to be integrated in the winner group.

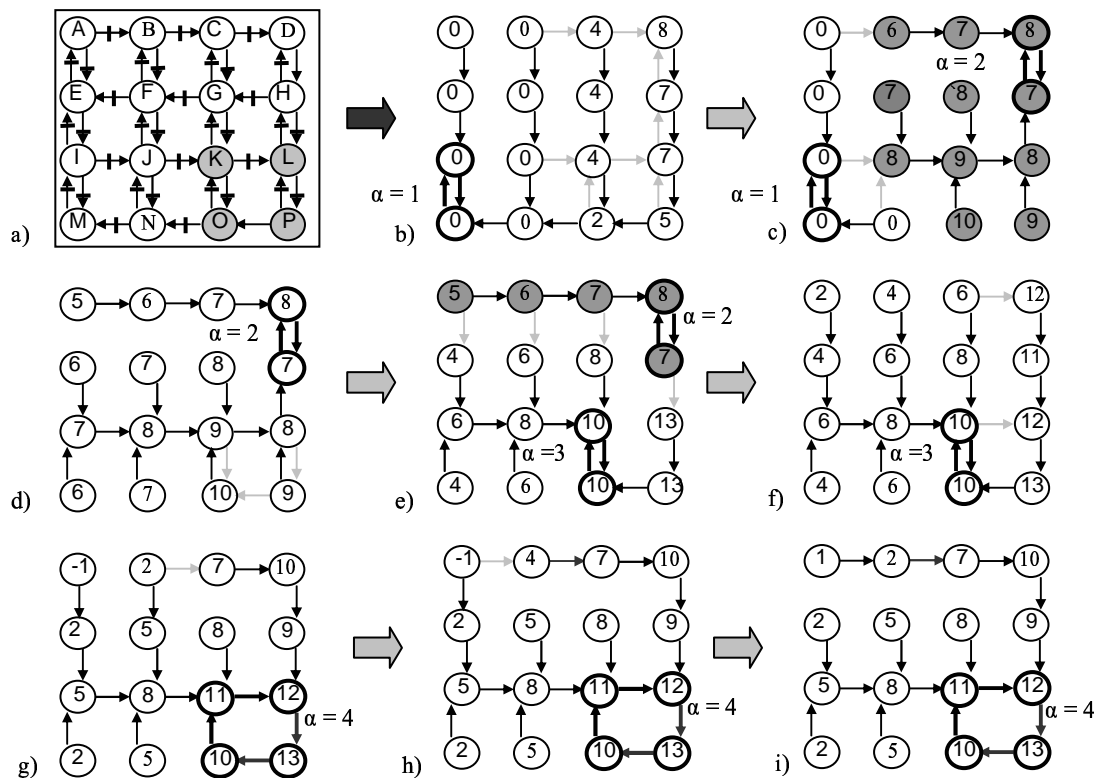


Figure 4. Social evolution of an HS scheduling algorithm example.

Following this process, all partitions are obtained until the critical circuit is found ( $f_2 = \alpha = 4$ ). This critical circuit has been obtained in the sixth partition (Figure 4.g), but the algorithm does not stop and continues the optimisation of the individual objective functions  $f_1$  and also it looks for new circuits, without success, in the following two partitions (seventh and eighth partitions). Finally, in the eighth partition (Figure 4.i), none of the nodes improves  $f_1$  or  $f_2$ , and the algorithm stops.

Once the critical circuit is obtained, the group and individual objective functions ( $f_2, f_1$ ) are used to schedule each task. Note that this is the first time in our knowledge that the retiming potential  $f_1$  is algorithmically used to directly determine the corresponding scheduling function.

To get a normalised scheduling, all the individuals objective functions are referred to the largest value. This normalisation can be made because the individual objective function is a potential function. In the

last partition, the final highest potential is associated to node P. This way, the scheduling  $S$  of each node  $u$  of the DFG will be given by the following expression:

$$S(u) = -f_1(u,i) + \max \{f_1(v,i)\} = -f_1(u,i) + 13$$

## 5. Improvements and Performance Analysis

We have introduced different options applied to the basic HS Scheduling algorithm to improve its performance. Note that the considered HS algorithm also computes the scheduling of each task when the iteration bound is being determined and no extra time is needed to compute the pipelining scheduling. For this reason no additional comparison is provided.

To have an estimation of the efficiency for the presented variants of HS scheduling algorithms and to evaluate their performance, all algorithms have been tested with some benchmarks. The benchmarks graphs used belong to three categories: I) graphs  $sxxx$  provide by ISCAS 89/93 benchmark [1]; II) graphs  $dsipx$  generated by SPRAND [1] building a Hamiltonian cycle and then adding arcs at random; III) several complex synthetic graphs  $abxxxx$  which were automatically generated by us. Note that ISCAS benchmarks and SPRAND graphs, commonly used in the literature to test algorithms for Profit-to-Time Ratio Problem, are not specifically oriented for testing critical circuits or scheduling optimization. In general, the out-degree of the graph nodes is relatively low, so there are few circuits and is relatively easy to solve the Maximum Profit to Time Ratio Problem [5]. For this reason, we have developed some synthetic graphs, which contain much more circuits.

### 5.1 Comparison among basic HS algorithm and other standard algorithms

The presented basic HS scheduling algorithm (HSA) is compared with: Karp and Orlin (KO), Young, Tarjan, and Orlin (YTO), Karp (K), Burns (B), Orlin and Ahuja (OA1), described in review [1].

Comparative execution times in seconds are shown in Figure 5 for some ISCAS and SPRAND benchmarks. The corresponding number of nodes and arcs ( $n$ ,  $m$ ) are also indicated. Notice that a logarithmic scale is used in the vertical axis.

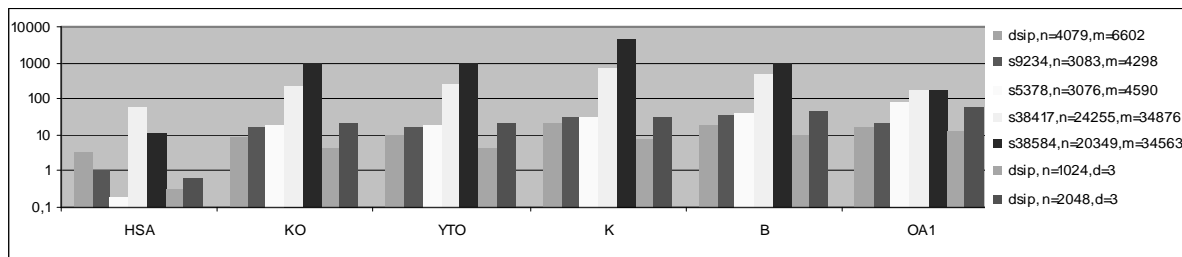


Figure 5. Comparison among basic HS scheduling algorithm and common used algorithms

### 5.2 Improvements of HS Scheduling algorithm

Once the relative good performance of the basic HS Scheduling algorithm is experimentally proved, several variants are considered to improve its computational performance. The best-improvement local search is used in all the variants studied.

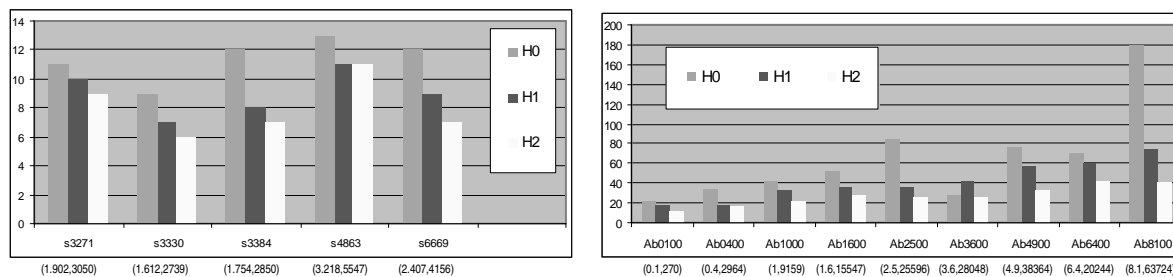
The main enhancement consists in the use of different autonomous behaviours of the groups. The tested variants are:

- **H0** basic implementation uses a simple *serial non-autonomous strategy* derived from Howard's algorithm [1], where the winner strategy, on the winner groups, is only applied when there are no loser groups.

- **H1** implementation uses a *semi-autonomous strategy* with a fixed number  $t$  of autonomous iterations, where each group independently evolves from the rest. Only the case ( $t=1$ ) is shown.
- **H2** implementation uses the standard *parallel non-autonomous strategy*, where in the same iteration the nodes belonging to loser and winner groups execute their corresponding strategy simultaneously, in order to improve the corresponding objective function.

Performed experiments are presented in Figure 6. Figure 6.a shows the number of iterations for some graphs *sxxxx* of ISCAS benchmark. These graphs are very simple in relation with the number of circuits. For this reason the algorithms were also tested with more complex benchmarks. The results of these experiments are shown in Figure 6.b for some developed benchmarks *abxxxx*. The corresponding number of nodes  $n$  and number of arcs  $m$  are shown in parenthesis ( $n, m$ ).

In general, for complex DFG's, the third implementation of the algorithm is faster than second one and the second implementation faster than the first one. Moreover, we can conclude that in most cases, more elaborated hierarchical social algorithms are more suitable for complex solution spaces with hard structure solutions. Notice that the second implementation is also highly suitable for graphs with very long circuits. In this situation, the semi-autonomous behaviour let to explore more deeply the possible critical circuits in a parallel form.



**Figure 6.** Comparison among variants (H0, H1, H2) of HS scheduling algorithm. The number of nodes, in parenthesis, is multiplied by a factor  $10^3$ .

## 6. Conclusion

In this paper we have presented hierarchical social algorithms to efficiently solve the optimisation problems of loop scheduling and software pipelining. First, we have highlighted a search algorithm as a social evolution process. The social paradigm described exploits the power of competition and cooperation to explore the solution space using a dynamical domain partition. Second, we have experimentally showed that the proposed HS Scheduling algorithms significantly reduce the number of iterations of the actually considered best one: Howard's algorithm. Furthermore, the use of more complex social metaphors also allows the natural extension of this metaheuristic.

## References

- [1] A. Dasdan, S. Irani, R.K Gupta (1998). An Experimental Study of Minimum Mean Cycle Algorithms. Technical report, UCI-ICS, 1998.
- [2] A. Duarte, F. Fernández, A. Sánchez, A. Sanz (2004). A Hierarchical Social Metaheuristic for the Max-Cut Problem. *LNCS*, Springer-Verlag, vol. 3004, pages 84-93, 2004.
- [3] A. Duarte, A. Sánchez, F. Fernández, A. Sanz, J.Pantrigo (2004). Top-Down Evolutionary Segmentation Using a HS Metaheuristic. *LNCS*, Springer-Verlag, vol. 3005, pages 301-311, 2004.
- [4] F. Fernández, A. Sánchez, A. Duarte. Software-Pipelining Method for Instruction-Level Parallel Processors based on Scaled Retiming (2001), In *Proceedings of the Conference ISPA2001*, vol.1, pages 86-92, Pula (Croatia), June, 2001.
- [5] T. O'Neil. Techniques for Optimizing Loop Scheduling. PhD thesis, Notre Dame, Indiana 2002.