A comparison between two optimisation alternatives for mapping in wireless network on chip

Maribell Sacanamboy
Universidad del Valle and
Pontificia Universidad Javeriana
Cali, Colombia
Email: msacanamboy@javerianacali.edu.co

Luis Quesada
Insight Centre for Data Analytics
University College Cork
Email: luis.quesada@insight-centre.org

Freddy Bolanos
Department of Electrical Eng. and Automatics
Universidad Nacional de Colombia - Medellín
Email: fbolanosm@unal.edu.co

Alvaro Bernal
Electrical and Electronics Eng. School
Universidad del Valle Colombia - Cali
Email: alvaro.bernal@correounivalle.edu.co

Barry O'Sullivan
Insight Centre for Data Analytics
University College Cork
Email: barry.osullivan@insight-centre.org

***Abstract*—Network on Chip (NoC) is a well known approach that aims at improving the performance of many-core systems. The design of such systems involves the optimal mapping of tasks to nodes, and the corresponding scheduling of the tasks at every node, which results in a challenging optimisation problem considering the constraints that need to be respected. In this paper, after formalising the problem and elaborating on its complexity, we present an AI approach to solve the problem and evaluate it against a MIP approach. Our empirical evaluation shows that the AI approach is able to obtain solutions of good quality very quickly.**

I. Introduction

Network on Chip (NoC) is a well-known approach that aims at improving the performance of many-core systems. NoC systems are composed by a set of processing elements or nodes, organised in a network, which implies the use of routers for the sake of sharing information among nodes. Such a network approach performs better with respect to traditional connection alternatives, such as buses. Generally, processors in the NoC are different from each other (i.e. the NoC is heterogeneous) which means that a clever computational load distribution may lead to very high performance implementations.

Nevertheless, as the amount of nodes grows in order to cope with performance constraints, the link latency (i.e. the time needed to transport messages among nodes) and the network traffic become a bottleneck in NoC systems [1]. As a result, Hierachical Network on Chip deals with higher network sizes, allowing the communication among a larger number of nodes. A Hierarchical NoC has several communication levels. Each level exhibits different features regarding link latencies and availability. Generally, one of these levels corresponds to wireless links, which is why such systems are also known as Wireless NoC or WNoC. The availability of several communication levels helps reduce mean link latency and cope with higher traffic profiles [1], [2].

Several proposals have been made concerning the hardware implementation of WNoC systems [3], [4]. In general, WNoC are composed by two communication levels. The first level corresponds to a wired, regular, mesh-type network, which communicates to adjacent nodes and configures subnets. When a given message must travel to a destination node which belongs to the same subnet such a message may use the first level links in order to reach its destination. However, if the message must travel between nodes that belong to different subnets, the second communication level must be used. This second level corresponds to a wireless, high-speed network.

The mapping and scheduling problem deals with the matching of tasks from an executable sentence with the available nodes and the execution order of tasks running on each node. When a given executable sentence is issued in the system, its tasks must be mapped to the WNoC nodes, in such a way that its execution optimises some measure of quality, e.g. elapsed time. There has been some work that deal with the mapping problem in NoC environments [5], [6], [7], but none of them refers to mapping in WNoC systems. Some other approaches deal with the efficient representation of both the sentence and the network architecture, for the sake of speeding up the mapping computation time [8], but does not consider the mapping problem directly.

Other approaches are able to issue mapping solutions online, i.e. at runtime. Such approaches allow the most complex computations to be performed offline, whereas a lighter algorithm may compute the mapping solution at runtime [9]. However, the reported results deal only with unrealistic network sizes which suggests that the this hybrid solution is not suitable for realworld applications. Other mapping efforts are devoted to the minimisation of power consumption [10], [11], which may be viewed as the quality metric to optimise, instead of the execution time of the application. However, such approaches use optimisation algorithms such as simulated annealing, which are very limited by the local-optimum drawback. We are aware of Benders decomposition approaches proposed for the mapping and scheduling problems [12]. However, the fact that the latency in our problem is node-dependent and multi-layer adds an extra level of complexity to our problem.

This paper describes a comparison between two modern optimisation approaches for the computation of mapping and scheduling of tasks. The quality metric to optimise has been chosen as execution time in a two-level WNoC, as described

earlier. The remainder of the paper is organised as follows. In Section II we present the mathematical model of the optimisation problem. In Section III we describe the PBIL algorithm. The experimental results are given in Section IV. Finally, Section V summarises our main conclusions and future work.

II. Mathematical Model of the Mapping Problem

We present the optimisation problem formally. We first describe the input of the problem and the set of variables. Then we introduce the constraints that need to be respected and the optimisation function. At the end we elaborate on both the complexity of the problem and the size of the mathematical model.

A. Input

- A dependency graph (T, L) on a given set of tasks T.
- A 2-dimensional matrix d representing the distance between each pair of nodes on a given network with set of nodes N. This distance has two components: one corresponding to the wired section of the path (d^w), and another corresponding to the wireless section of the path (d^r).
- A 2-dimensional matrix t representing the time that the task would spend if executed at a given node. That is, t^i_a is the time that task i spends if executed at node a.
- A set of incompatible pairs of tasks P. In what follows we will often call P the incompatibility graph.
- Each router in the networks has its own latency. In the model we use λ^w to denote the wired latency, and λ^r to denote the wireless one.
- The speed of the link depends on whether it is wired or wireless. σ_w refers to the speed of a wired link and σ_r to the speed of a wireless one.
- The amount of data that task i sends to task j is denoted by π_{ij}.

B. Variables

- A Boolean variable x^i_a denoting whether task i is placed at node a.
- A Boolean variable y^{ij}_{ab} denoting whether task i is placed at node a and task j is placed at node b.
- A Boolean variable p^{ij} to denote whether task i is using a wired link when going from i to j.
- A Boolean variable q^{ij} to denote whether task i is using a wireless link when going from i to j.
- A Boolean variable ψ^{ij} to denote whether task i precedes task j.
- A float variable δ^{ij}_w denoting the wired distance from the node of task i to the node of task j.
- A float variable δ^{ij}_r denoting the wireless distance from the node of task i to the node of task j.
- A float variable τ^i denoting the execution time of task i.
- A float variable s^i denoting the time at which task i is starting.
- A float variable ω denoting total execution time (makespan).

C. Constraints

1) The distance from the node of task i to the node of task j will depend on the nodes associated with the tasks. Note that there is only one variable y^{ij}_{ab} that will be set to 1 since each task can only be mapped to one node. The following constraint will ensure that δ^{ij}_w and δ^{ij}_r take the expected value based on the assignment of tasks to nodes:

$$\forall \langle i,j \rangle \in L : \delta^{ij}_w = \sum_{a \in N, b \in N} d^w_{ab} \times y^{ij}_{ab} \tag{1}$$

$$\forall \langle i,j \rangle \in L : \delta^{ij}_r = \sum_{a \in N, b \in N} d^r_{ab} \times y^{ij}_{ab} \tag{2}$$

2) We define p and q accordingly. Notice that $(d^w_{ab} > 0)$ and $(d^r_{ab} > 0)$ are Boolean constants:

$$\forall \langle i,j \rangle \in L : p^{ij} = \sum_{a \in N, b \in N} (d^w_{ab} > 0) \times y^{ij}_{ab} \tag{3}$$

$$\forall \langle i,j \rangle \in L : q^{ij} = \sum_{a \in N, b \in N} (d^r_{ab} > 0) \times y^{ij}_{ab} \tag{4}$$

3) We define the y variables in term of the x variables in the expected way:

$$\forall \langle i,j \rangle \in L, a \in N, b \in N : y^{ij}_{ab} \Leftrightarrow x^i_a \wedge x^j_b \tag{5}$$

4) Each task is assigned to one node only:

$$\forall i \in T : \sum_{a \in N} x^i_a = 1 \tag{6}$$

5) The tasks in an incompatible pair of tasks should be placed in different nodes:

$$\forall (i,j) \in P, a \in N : x^i_a + x^j_a < 2 \tag{7}$$

6) The execution time of a task depends on the node the task is executed:

$$\forall i \in T : \tau_i = \sum_{a \in N} t^i_a \times x^i_a \tag{8}$$

7) If task j depends on task i, then task j has to wait until receiving the data from task i:

$$\begin{aligned} \forall \langle i,j \rangle \in L : s^j \geq \; & s^i + \\ & \tau^i + \\ & (\pi^{ij}/\sigma_w) \times \delta^{ij}_w + \\ & (\pi^{ij}/\sigma_r) \times \delta^{ij}_r + \\ & \lambda^w \times p_{ij} + \\ & \lambda^r \times q_{ij} \end{aligned} \tag{9}$$

8) The end of execution cannot be reached before the end of a task:

$$\forall i \in T : s^i + \tau^i \leq \omega \tag{10}$$

9) If task i precedes task j, task j must wait for task i:

$$\forall i,j \in T : s^j \geq s^i + \tau^i - \alpha \times (1 - \psi_{ij}) \tag{11}$$

where α is a constant greater than the maximum start time possible for task j. That is, if task i is not constrained to precede task j, then the previous inequality is a tautology.

10) If task j depends on task i, task i precedes task j:

$$\forall \langle i,j \rangle \in L : \psi_{ij} = 1 \tag{12}$$

11) If task i and task j are executed by the same node, one of them must precede the other one:

$$\forall i,j \in T, a \in N : (x_a^i \wedge x_a^j) \Rightarrow (\psi_{ij} \vee \psi_{ji}) \tag{13}$$

12) A node can only handle one task at a time, so ω is greater than or equal to the sum the execution times of the tasks allocated to it:

$$\forall a \in N : \omega \geq \sum_{i \in T} t_a^i \times x_a^i \tag{14}$$

This is a redundant constraint that is quite useful to tighten the lower bound of ω.

$y_{ab}^{ij} \Leftrightarrow x_a^i \wedge x_b^j$ is not really an integer linear equation. However it can be translated into a set of linear equations by putting the Boolean formula in conjunctive normal form and translating each clause into its corresponding linear equation formula. That is:

$$y_{ab}^{ij} \Leftrightarrow x_a^i \wedge x_b^j \equiv (\neg x_a^i \vee \neg x_b^j \vee y_{ab}^{ij}) \wedge (x_a^i \vee \neg y_{ab}^{ij}) \wedge (x_b^j \vee \neg y_{ab}^{ij})$$

Once we have the formula in conjunctive normal form, it can be easily translated into an equivalent set of linear equations as follows:

$$\begin{aligned} x_a^i + x_b^j - y_{ab}^{ij} &< 2 \\ x_a^i - y_{ab}^{ij} &> -1 \\ x_b^j - y_{ab}^{ij} &> -1 \end{aligned}$$

Similarly, $(x_a^i \wedge x_a^j) \Rightarrow (\psi_{ij} \vee \psi_{ji})$ is not a linear equation either. However, in this case the transformation is straightforward since it is equivalent to $\neg x_a^i \vee \neg x_a^j \vee \psi_{ij} \vee \psi_{ji}$.

D. Objective

The objective is to minimise the makespan (ω).

E. Complexity of the optimisation problem

We recall that for each task we are deciding the node on which the task will be executed and the time at which the task is executed. Even if we focus on the mapping problem only, it is not difficult to prove that the optimisation problem we are addressing is NP-Hard. We can reduce the well known NP-Hard Graph Colouring Problem [13] to our problem as follows:

- The set of nodes in the WNoC system represents the set of colours available.
- The set of tasks corresponds to the set of nodes of the graph.
- Each incompatibility in the incompatibility graph represents an arc in the graph that we want to coulor.

Then, the question of whether we can colour a graph with m colours is equivalent to the question of whether we can find a mapping of tasks to nodes assuming we have m nodes available for executing the tasks.

F. Size of the mathematical model

The size of the mathematical model is dominated by the number of linear equations, which depends on the number of tasks in T, the number of nodes in N, the number of dependencies in L, and the number of incompatibilities in P. We observe that the constraint leading to the highest number of linear equations is Constraint 3. If we assume that the number of tasks is comparable to the number of nodes (i.e., $|N| \sim |T|$), and the dependency graph is dense (i.e., $|L| \sim |T|^2$), this constraint would lead to $\mathcal{O}(|T|^4)$ linear equations.

III. The PBIL Algorithm

A. Description of the algorithm

PBIL is a heuristic optimisation algorithm inspired by evolutionary algorithms and also from competitive learning neural networks [14]. As its name suggests, PBIL is an algorithm that works on a population of potential solutions to a given optimisation problem. Any combinatorial problem can be represented with a set of attributes. Each attribute may have a set of potential values depending on the problem at hand. The PBIL approach mainly relies on a probability array that stores the search information that has been acquired so far. Figure 1 shows an example of a PBIL probability matrix (B) for a generic problem with V attributes, for which there may be up to U choices for each attribute. In the specific context of WNoC mapping problems, attributes refer to tasks and the choices refer to the nodes where such tasks may be executed.

In Figure 1, $B(i,j)$ stands for the probability of attribute j taking value i. As the algorithm converges (through search iterations), some of the probabilities of the array become

	Atributte 1	Atributte 2	...	Atributte V
Choice 1	B(1,1)	B(1,2)	...	B(1,V)
Choice 2	B(2,1)	B(2,2)	...	B(2,V)
...	...	...	...	...
Choice U	B(U,1)	B(U,2)	...	B(U,V)

Fig. 1. A typical PBIL probability array

Input: $Tolerance$, A dependency graph (T, L) on a given set of tasks T, An architectural graph of the WNoC, B {The PBIL probability array}
Output: $Optimal\_solution$
1: $B(i,j) \leftarrow 1/|U| \quad \forall \quad 1 \leq i \leq |U|$ **and** $1 \leq j \leq |V|$
2: **repeat**
3: $Population \leftarrow Generate\_Population(B)$;
4: $Best \leftarrow Choose\_Best(Population)$;
5: $Entropy \leftarrow Assess(B)$;
6: $LR \leftarrow Learning\_Rule(Entropy)$;
7: $B \leftarrow Update(B, Best, LR)$;
8: **until** $(Entropy \leq Tolerance)$
9: $Optimal\_solution \leftarrow Best$;
Algorithm 1: A basic Adaptive PBIL algorithm

higher than others, which means that those choices are related to better solutions in the search space. At the beginning of the algorithm it is common to have the whole set of probabilities assigned to the same value, which means that all the choices are equally probable in early stages of the search process. In constrained scenarios, where some choices are precluded, the associated probabilities are set to zero. Algorithm 1 shows the iterative convergence process for a typical PBIL optimisation approach.

Line 1 in Algorithm 1 implements the initialisation process for the probability array (B). Since there are up to $|U|$ choices for each attribute, as can be observed from Figure 1, each probability in array B is set to $1/|U|$. This means that each solution of the search space has the same probability as the remaining ones when the algorithm begins. In Line 3 we generate a new population by calling $Create\_Population$. Those choices with higher probabilities will be more prone to be chosen in such generated population.

Once the population is generated, the best solution of such population must be found, for the sake of guiding the search process. The $Choose\_Best$ routine locates the best solution in the population, by using as criterium the figure of merit concerning the optimisation process. The $Choose\_Best$ routine is located at Line 4 in Algorithm 1. At the same time it is necessary to assess the convergence state of the algorithm. This is performed by a measure of the entropy of array B, as shown in Line 5 of the algorithm. The entropy calculation is performed by means of the classical Shannon's formulation for the probabilities in array B [15].

The learning rate parameter is represented by the LR variable in Algorithm 1 (Line 6). This parameter serves to adjust the convergence speed of the algorithm. Higher values of LR result in faster convergence times, at the expense of poorer solution quality. This is referred to as exploitation of the search space. Lower values of LR lead to solution space exploration, i.e. slower searching for the sake of improving the solution quality. An adaptive approach of the PBIL algorithm adjusts the LR parameter dynamically in order to favour the exploration at early stages of the convergence process and improve exploitation at the end of such convergence [7]. The way in which the LR parameter changes is referred to as the *learning rule*. A *linear learning rule* was implemented in the $Learning\_Rule$ routine, at Line 6 in Algorithm 1 [5].

Once the LR parameter is ready it is time to update the probabilities in the B array in order to approach the best solution found at the current iteration. This task is performed by the $Update$ routine, which must increase the probabilities associated with the best choice for each attribute. Since an attribute may be viewed as a complete probability event, the sum along each column of the probability array in Figure 1 must be equal to one. As a consequence, if a single probability is increased (i.e. the one associated to the best solution) the remaining probabilities in this column must be decreased accordingly. The $Update$ routine performs changes to the probabilities by means of a modified hebbian rule, as depicted in Equation (15).

$$B(i,j)_N = \begin{cases} B(i,j) + \left[1 - B(i,j)\right] \cdot LR, & \text{if } j = k \\ \frac{\left[1 - B(i,k)_N\right] \cdot B(i,j)}{1 - B(i,j)}, & \text{otherwise.} \end{cases} \quad (15)$$

In Equation (15), it is assumed that the best choice for attribute i corresponds to the k-th position, which means that such an entry in the matrix P must be increased. The remaining probabilities in column i must be decreased proportionally. $B(i,j)_N$ refers to the updated value of the probability at entry (i,j), whereas $B(i,j)$ refers to the value prior to the updating process.

This iterative process – which involves the generation of a population starting from the B array, the search of the best solution or makespan, and the adjusting of the original probability values according to such best solution – repeats itself until the *Entropy* becomes low enough. As the probabilities tend to concentrate in some individual positions of the array, the *Entropy* tends to zero. The termination of the convergence process compares *Entropy* values with a $Tolerance$, since reaching zero is a very restrictive condition. At the end, the best solution found so far is returned as the best makespan.

The basic version of the PBIL algorithm depicted in Algorithm 1 must be customised at some key points to solve the mapping of tasks in WNoC environments, as described previously. Such customisations deal in the first place with the probability array. Figure 1 shows a typical probability matrix where there are a set of V attributes for which there may be up to U implementation choices. The representation of the mapping problem is done by associating each attribute in the PBIL algorithm with a task to be executed. Consequently, as each attribute is associated with an executable task, and the optimisation problem deals with the best combination of resources (nodes) for implementing such tasks, a mapping solution is completely defined by a set of resources, V, which represents the set of tasks of the input instance. In other words, it can be said that for Figure 1 V corresponds to the number of tasks of the input instance and U corresponds to the set of resources that are suitable to implement such tasks (i.e. $V = T$ and $U = N$).

At the end of the optimisation process, since probabilities are concentrated in some specific entries of the matrix – for each column in the array, there will be a entry with a probability value above the remaining ones – the best solution may be easily derived by simply taking such options with higher probability, representing implementation resources, for each attribute, i.e. each task. If there is a constraint regarding some specific task-resource combination, for instance it is precluded that some task to be executed in some specific node, the initialisation process (Line 1, in Algorithm 1) must set the associated probability of such a combination to zero so that none of the subsequent generated populations take into account such a choice. This condition implies that, for that specific column, the initialisation of the remaining probabilities should be set to $1/(|U|-1)$.

We ensure that the constraints of the problem are met in the implementation of $Generate\_Population$. This is done by discarding those members of the population that do not meet the constraints so that they are not taken into account when choosing the best solution, and the probabilities in the PBIL array are not affected by those members.

B. Complexity of the algorithm

The complexity of the algorithm is given in terms of the input of the problem. In what follows we focus on the complexity of the five instructions comprising the loop.

- $Generate\_Population$. As already mentioned, this procedure is responsible for creating a population of potential solutions. This population is represented by an array of dimensions $|Ps|$ by $|T|$, where $|Ps|$ corresponds to the population size, and $|T|$ corresponds to the number of tasks in the input instance. This function is implemented by means of two nested loops, whose limits of iteration are $|Ps|$ and $|T|$. In the inner loop the validation for each matrix population solution is performed. This validation is done by two nested cycles with an upper limit of iteration $|N|-1$, the algorithm assigns the first task to a node randomly, validating the assignment based on the incompatibility graph. The number of operations required for $Generate\_Population$ routine is $\mathcal{O}(|Ps| \times |T| \times [(|N|-1)^2 + |P|])$.
- $Choose\_Best$. This function evaluates the population matrix based on the optimisation criterion and determines the makespan evaluating a vector containing the best solutions. This routine is based on a cycle whose upper limit is $|Ps|$. Within this cycle two functions are called. The first function calculates the latencies for all tasks for each solution of the population matrix and it is defined by two nested cycles, which have an upper limit of $|T|$. The second function computes a vector with the best solutions, calculating the node time and communication channels for all paths in the dependency graph. These times are stored in a matrix. This array is filled by two nested loops bounded by $|L|$ and $|N|^2$. Finally, a vector is obtained with the best solutions through a cycle, in which each row of the matrix of paths is added. The complexity of this final loop is $\mathcal{O}(|L|)$. The total number of operations required by $Choose\_Best$ is: $\mathcal{O}(|Ps| \times [|T|^2 + (|L| \times |N|^2) + |L|])$.
- $Assess$. This function computes the Shannon entropy for the population matrix using the sum instruction of Matlab, which is an operator that adds up all the data of the array B, so the complexity of this step is $\mathcal{O}(|N| \times |T|)$.
- $Learning\_Rule$. This function computes the learning rate parameter and is based on the value of entropy (E) of B. This parameter is obtained by applying a linear learning rule [5], which is represented by the linear equation $LR = mE + b$. m and b are adjusted taking into account the minimum and maximum values of LR (LR_{min}, LR_{max}), which are part of the input of the algorithm. The complexity of $Learning\_Rule$ is $\mathcal{O}(1)$ since it is independent of the size of the input of the problem.
- $Update$. This routine updates the probability matrix using two nested loops with limits of iteration $|T|$ and $|N|$, so the complexity of this step is $\mathcal{O}(|T| \times |N|)$.

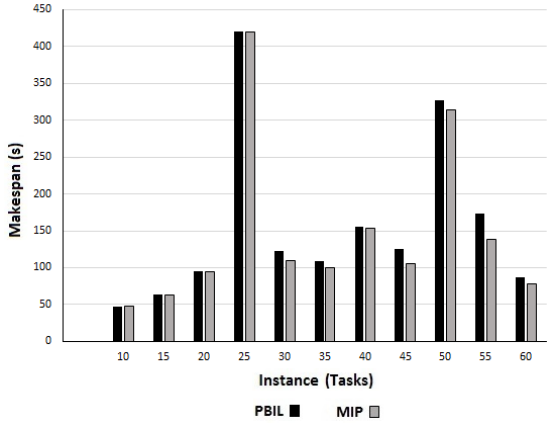
According to the above, $Choose\_Best$ has the highest complexity, so it has the strongest impact on the performance of the PBIL algorithm.

IV. Experimental Results

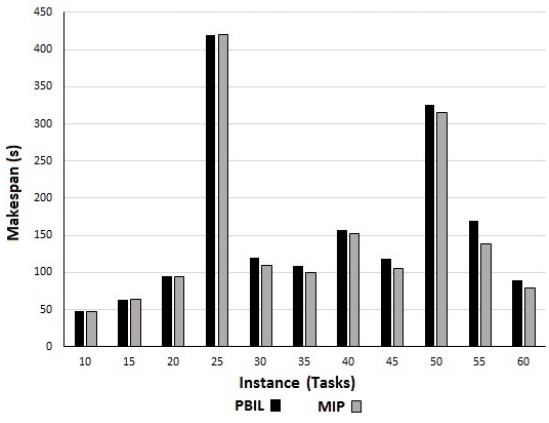
The tests were realised using a set of eleven synthetic instances generated through the Graph Tool called TGFF (Task Graphs For Free) [16], which is a GNU tool used for embedded system applications. These instances consist of tasks, which are described by one task graph. In this case from ten to sixty tasks with increments of five tasks were considered. The TGFF tool provides the profile of both bandwidth and execution times of processors for each task in the instance.

The hardware used to solve the problem of task assignment was modelled using a hierarchical wireless heterogeneous two-level architecture. The first level consists of four subnets of four nodes per subnet. The topology of these subnets is mesh-like and each subnet is wiredly interconnected. The second level is determined by a star topology and allows wireless interconnection between all subnets placed in the first level. The nodes in each subnet were composed of two elements: the first one related to processing features determined by a processor with different technological characteristics and the second one related to communication features defined by a router. The network traffic is modelled by a simple XY routing algorithm. The algorithm PBIL was implemented and tested using the R2016 tool Matlab. The MIP was implemented in CPLEX using its Python interface [17].

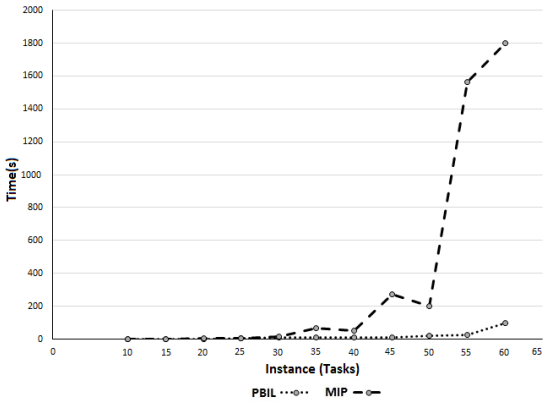
As explained in Section II, as part of the input we have the dependency graph and the incompatibility graph. In Table I we show the number of edges in columns L and P respectively. The number of nodes for all these instances is 16. We show the dependency graph and the incompatibility graph of the instance involving 20 tasks in Figures 3 and 4. As it can be observed, the dependency graph has long dependency paths



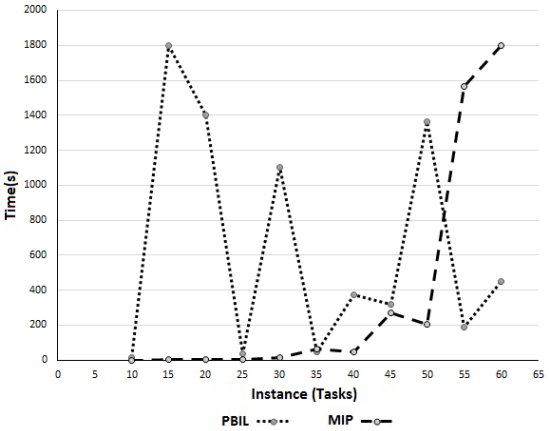
(a) Best solution found with the MIP and PBIL optimisation approaches (allowing the PBIL approach to iterate 10 times)



(b) Best solution found with the MIP and PBIL optimisation approaches (allowing the PBIL approach to iterate until it finds the best solution)



(c) Processing Time to find the best solution for both optimisation approaches MIP and PBIL (allowing the PBIL approach to iterate 10 times)



(d) Processing Time to find the best solution for both optimisation approaches MIP and PBIL (allowing the PBIL approach to iterate until it finds the best solution)

Fig. 2. Best solutions and execution times of the two optimisation approaches.

like the one going from task 1 to task 17, which contains tasks 1, 4, 7, 12, 16, and 17. These dependencies impose a total order on the execution of these tasks, i.e. the makespan is bound to be greater than or equal to the sum of the times of execution of those tasks. In the incompatibility graph we observe that we have four components. All components are actually cliques which leads us to conclude that at least m nodes would be required, where m is the size of the biggest clique. Notice, however, that all the tasks in the dependency path mentioned above can be executed in the same node since there is no incompatibility constraint between them.

As mentioned before, the optimisation criterion for these tests was the minimum time of execution (makespan) obtained from the optimal mapping of tasks to nodes and the corresponding scheduling of the tasks. For both optimisation alternatives, a stopping criterion timeout of 1800 seconds was considered.

In Figure 2, we show the results obtained for the eleven instances by the two optimisation alternatives. In Figure 2(a) and 2(b) the Y-axis refers to the makespan given in seconds and the X-axis shows the eleven instances represented by the number of tasks, ranging from ten to sixty tasks. In Figure 2(c) and 2(d) the Y-axis refers to the execution time given in seconds.

Recall that the PBIL approach is random. So, in order to evaluate its performance we have run the procedure several times. In Figures 2(a) and 2(c) we have run the procedure a fixed number of times (10 times) per instance and compared the best solution with the one obtained by the MIP approach. As we can observe, the solutions obtained are very close in quality with respect to the ones obtained by the MIP approach. The time spent by the PBIL approach is very low if we take into account the time spent by the MIP approach. In Table I we also report the maximum value and the median value over the 10 iterations, and compare that with the actual values obtained by the MIP approach.

In Figures 2(b) and 2(d) we have allowed the PBIL approach to run as many times as possible within a timeout of

TABLE I
MAKESPAN AND TIME FOR PBIL AND MIP

Instance			Makespan(s)			Time(s)			**Makespan(s)**	**Time(s)**	**Gap**
T	L	P	Min	Max	$Median$	Min	Max	$Median$	MIP	MIP	
10	9	10	47.43	54.17	51.48	0.70	1.49	0.92	47.41	0.79	0.04
15	16	20	63.33	70.16	68.00	1.36	2.21	1.63	63.24	1.72	0.14
20	24	26	94.45	105.50	104.00	2.45	3.34	2.77	94.45	3.79	0.00
25	32	41	420.09	434.58	423.29	3.32	6.53	4.11	420.09	4.42	0.00
30	42	53	122.76	134.93	131.49	6.16	12.42	7.65	109.17	17.40	12.45
35	49	63	109.31	127.02	117.76	8.98	14.89	10.62	99.85	65.08	9.48
40	48	115	155.15	169.72	165.28	6.96	54.78	10.12	152.92	49.63	1.46
45	54	141	125.97	143.72	134.46	8.72	55.86	11.38	105.76	270.74	19.11
50	73	107	327.41	353.16	340.47	20.81	28.69	23.21	314.66	202.47	4.05
55	78	117	173.28	208.69	188.31	20.58	41.40	26.03	139.02	1563.26	24.64
60	73	264	86.73	100.32	94.38	19.16	237.51	55.07	78.50	1800.00	10.48

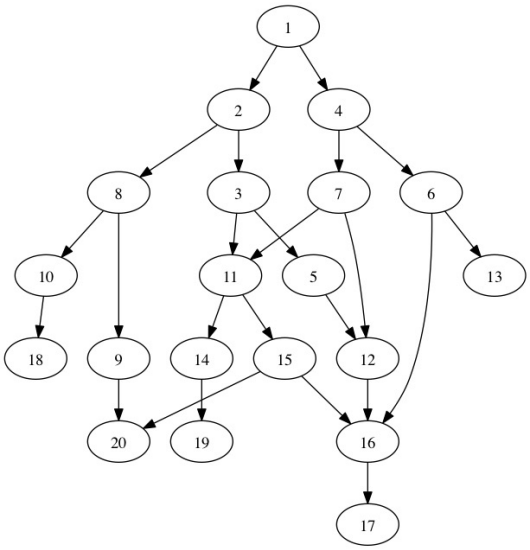


Fig. 3. Dependency graph for the instance of 20 tasks

30 minutes. The time reported, though, is the time at which the best solution is found. We observe a small improvement in the quality of the solutions but the number of iterations required to reach such solution is high (as it can be observed in Figure 5), which leads to considerably higher execution times.

In Table I we also show the gap between the best makespan obtained by PBIL after 10 iterations and the one obtained by the MIP approach. This gap does not evolve monotonically with respect to the number of tasks in the instance. This is due to the fact that the complexity of the dependency graph and the incompatibility graph does not increase monotonically with respect to the number of tasks either. Another thing worth noting is that the MIP approach does not scale. Indeed we observe that the execution time increases considerably after 50 tasks. However, it is important to remember that the MIP approach is proving optimality, which means that a suboptimal solution of decent quality can be computed in much less time.

V. CONCLUSION

MIP is a deterministic optimisation method that ensures optimality in all cases. On the other hand PBIL is a heuristic approach that does not guarantee that the best solution or makespan from the search space is found. The reason for this is because PBIL works with a stochastic search based on estimates and randomness, so there is no certainty on finding the global optimum.

However, in our experiments we showed that the PBIL approach obtains solutions that are very close to the optimal ones using a small amount of time. While the MIP approach guarantees optimality, it does not scale to instances involving a large number of tasks, not only because it takes considerably more time to prove optimality, but also because the size of its model increases quite rapidly with respect to the number of nodes and tasks.

As noted earlier, we are aware of Benders decomposition approaches proposed for related mapping and scheduling problems [12]. We plan to build on those approaches to develop a more scalable complete approach taking into account that one of the challenges in our case is to model the node-dependent multi-layer latency in the master problem.

We have focussed on the optimisation problem associated with the mapping and scheduling of tasks. While this problem is interesting in its own right, we have observed that the user usually finds it hard to understand why it is not possible to achieve some expected performance. Motivated by this fact, as part of our future work, we also plan to implement a tool that supports the user in this task. More concretely, given an infeasible target (makespan) our objective is to come up with a preferred explanation for infeasibility. In Section II we presented the mathematical model containing 12 types of constraints. There are certainly constraints that are more relevant than others when it comes to explaining the reason for infeasibility. For instance, the user may want to be presented first we dependency constraints and even between dependencies there may be some dependencies that are more important than others.

ACKNOWLEDGMENT

The authors would like to thank Pontificia Universidad Javeriana Cali, Universidad Nacional de Colombia and Universidad del Valle for their support in the development of the current paper. This work is partially supported by the Science Foundation Ireland under Grant No.10/CE/I1853. The

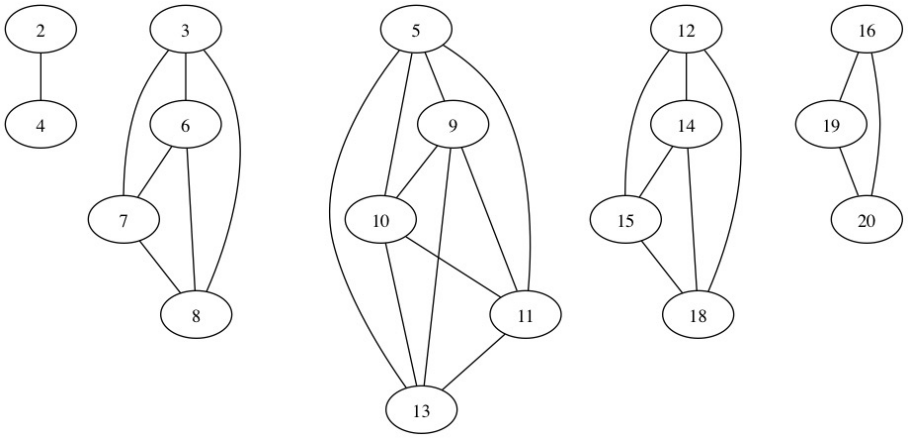


Fig. 4. Incompatibility graph for the instance of 20 tasks

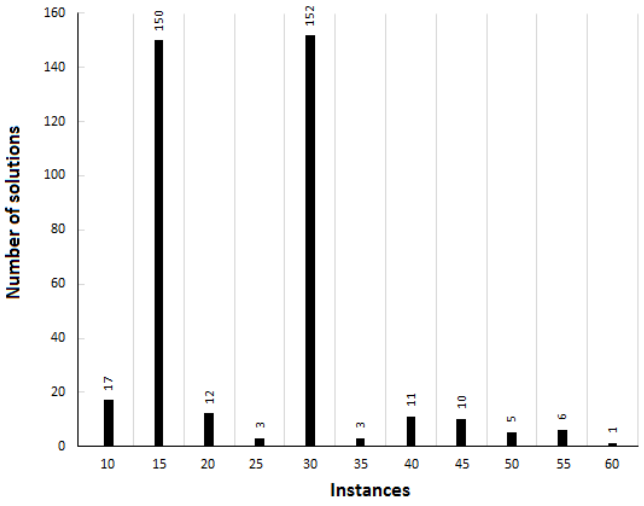


Fig. 5. Number of attempts carried out by PBIL to find the best solution

Insight Centre for Data Analytics is also supported by Science Foundation Ireland under Grant No. SFI/12/RC/2289.

REFERENCES

[1] A. Guerre, N. Ventroux, R. David, and A. Merigot, "Hierarchical network-on-chip for embedded many-core architectures," *NOCS 2010 - The 4th ACM/IEEE International Symposium on Networks-on-Chip*, pp. 189–196, 2010.

[2] A. Lankes, T. Wild, and A. Herkersdorf, "Hierarchical NoCs for optimized access to shared memory and IO resources," *12th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, DSD 2009*, pp. 255–262, 2009.

[3] X. Li, *Survey of Wireless Network-on-Chip Systems*. PhD thesis, Auburn University, 2012.

[4] A. Rezaei, F. Safaei, M. Daneshtalab, and H. Tenhunen, "HiWA : A Hierarchical Wireless Network-on-Chip Architecture," *International Conference on High Performance Computing & Simulation, HPCS*, pp. 499–505, 2014.

[5] F. Bolanos, J. E. Aedo, F. Rivera, and N. Bagherzadeh, "Mapping and Scheduling in Heterogeneous NoC through Population-Based Incremental Learning," *Journal of Universal Computer Science*, vol. 18, no. 7, pp. 901–916, 2012.

[6] F. Bolaños, *Mapping Techniques for Embedded Systems Design with Reliability Considerations*. PhD thesis, University of Antioquia Medellín, Colombia, 2012.

[7] F. Bolaños, J. Aedo, and F. Rivera, "Static and dynamic task mapping onto network on chip multiprocessors," *Dyna*, vol. 81, no. 185, pp. 28–35, 2014.

[8] P. M. Wells, K. Chakraborty, and G. S. Sohi, "Adapting to intermittent faults in multicore systems," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 255–264, Mar. 2008.

[9] A. K. Singh, A. Kumar, and T. Srikanthan, "A hybrid strategy for mapping multiple throughput-constrained applications on mpsocs," in *Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '11, (New York, NY, USA), pp. 175–184, ACM, 2011.

[10] C. Çelik and C. F. Bazlamaçci, "Effect of application mapping on network-on-chip performance," in *Proceedings of the 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2012, Munich, Germany, February 15-17, 2012*, pp. 465–472, 2012.

[11] E. Antunes, M. Soares, A. Aguiar, S. J. Filho, M. Sartori, F. Hessel, and C. A. M. Marcon, "Partitioning and dynamic mapping evaluation for energy consumption minimization on noc-based mpsoc," in *Thirteenth International Symposium on Quality Electronic Design, ISQED 2012, Santa Clara, CA, USA, March 19-21, 2012*, pp. 451–457, 2012.

[12] A. Emeretlis, G. Theodoridis, P. Alefragis, and N. Voros, "A logic-based benders decomposition approach for mapping applications on heterogeneous multicore platforms," *ACM Trans. Embed. Comput. Syst.*, vol. 15, pp. 19:1–19:28, Feb. 2016.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[14] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," tech. rep., Pittsburgh, PA, USA, 1994.

[15] C. E. Shannon, "A mathematical theory of communication," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, pp. 3–55, Jan. 2001.

[16] R. Dick and D. Rhodes, "Task Graphs For Free (TGFF)." Available at:http://ziyang.eecs.umich.edu/ dickrp/tgff/, 1998.

[17] IBM ILOG CPLEX Optimization Studio, "http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud," April 15 2016.