Static routing for applications mapped on NoC platform using ant colony algorithms

Luneque Del Rio de Souza e Silva Junior* and Nadia Nedjah

Department of Electronics Engineering and Telecommunication, Faculty of Engineering, State University of Rio de Janeiro, Rua São Francisco Xavier, 524, Maracanã, Rio de Janeiro, Brazil
E-mail: luneque@hotmail.com
E-mail: nadia@eng.uerj.br
*Corresponding author

Luiza de Macedo Mourelle

Department of Systems Engineering and Computation, Faculty of Engineering, State University of Rio de Janeiro, Rua São Francisco Xavier, 524, Maracanã, Rio de Janeiro, Brazil
E-mail: ldmm@eng.uerj.br

Abstract: Networks-on-chip (NoC) have been used as an interesting option in design of communication infrastructures for embedded systems, providing a scalable structure and balancing the communication between cores. Because several data packets can be transmitted simultaneously through the network, an efficient routing strategy must be used in order to avoid congestion delays. In this paper, ant colony algorithms were used to find and optimise routes in a mesh-based NoC. The routing optimisation is driven by the minimisation of total latency in packets transmission between tasks. The simulation results show the effectiveness of the ant colony inspired routing by comparing it with general purpose algorithms for deadlock free routing.

Keywords: network-on-chip; NoC; routing algorithms; packet routing; ant colony optimisation; ACO; application task graphs.


Biographical notes: Luneque Del Rio de Souza e Silva Junior is a graduate student. He received his Engineering degree in Electric Engineering from State University of Rio de Janeiro, Brazil. He is pursuing his Master in Electronics Engineering and his interests are swarm intelligence in particular and for intelligent systems in general.

Nadia Nedjah is an Associate Professor in the Department of Electronics Engineering and Telecommunications at the Faculty of Engineering, State University of Rio de Janeiro, Brazil. Her research interests include functional programming, embedded systems and reconfigurable hardware design as well as cryptography. She received her PhD in Computation from the University of Manchester – Institute of Science and Technology (UMIST), England, MSc in System Engineering and Computation from the University of Annaba, Algeria and Engineering degree in Computer Science also from the University of Annaba, Algeria.

Luiza de Macedo Mourelle is an Associate Professor in the Department of System Engineering and Computation at the Faculty of Engineering, State University of Rio de Janeiro, Brazil. Her research interests include computer architecture, embedded systems design, hardware/software codesign and reconfigurable hardware. She received her PhD in Computation from the University of Manchester – Institute of Science and Technology (UMIST), England, MSc in System Engineering and Computation from the Federal University of Rio de Janeiro (UFRJ), Brazil and Engineering degree in Electronics also from UFRJ, Brazil.
1 Introduction

A system-on-chip (SoC) is an integrated circuit composed by a full computer system. SoCs contains, within the same package, processors, memory, input-output controllers and specific application devices. This block structure follows a design methodology based on intellectual property (IP) cores. The increase of SoC’s scale raises new design challenges. Among them is communication between IP cores. For many SoC designs, it is desirable to use a communication framework scalable and as fast. An architecture that includes these two features is networks-on-chip (NoC) (Benini and De Micheli, 2002).

In a NoC architecture, switches are interconnected by point-to-point links, thus describing a network topology. The switches are also connected to the IP cores that constitute the system, also called resources. Switches exchange information in the form of messages and packages. This operation is similar to that performed by computer networks. NoCs can be used in the implementation of multi-processors systems-on-chip (MPSoCs) for running applications with high level of parallelism (Moureelle et al., 2010).

In the design of embedded systems, the communication plays an important role. To assist the designer, computational tools for project, or electronic design automation (EDA), are used (Józwicki et al., 2010). The purpose of EDAs is to optimise intermediate stages of NoC projects, in order to obtain a more efficient design implementation quickly. The EDA tools must be able to use information about the desired application (at a high level of abstraction) and, through successive stages of optimisation, implement a solution that meets the design specification. This optimisation may include several steps, such as task allocation (Da Silva et al., 2009b), IP mapping (Nedjah et al., 2011) and static routing. Figure 1 shows in a simplified way the flowchart of a SoC design based on NoC.

Figure 1 Typical embedded system design flow for NoC platform

In a NoC architecture, delays in communication may occur in congestion situations, when multiple packets could be transmitted using the same switch at the same time. In order to overcome this congestion problem, and thus accelerating the packet delivery, which, in turn, would allow for an improvement of the whole execution time of the system, this paper proposes a route optimisation step in the design of NoCs, or more precisely, an adaptive and static routing. In this paper, the algorithm used in the search for routes is the ant colony optimisation (ACO) (Dorigo et al., 1996).

The reminder of this paper is organised as follows. In Section 2, we review the related work in routing algorithms. In Section 3, we do an overview on ACO meta-heuristics. The proposed routing is presented in Section 4. A brief description of applications and mapping is shown in Section 5. Simulation results are presented in Section 6. This paper closes with a conclusion and a description of future work in Section 7.

2 Network routing

There are several works that study the efficient routing in parallel and distributed computing. For a broader reference, Ni and McKinley (1993) present a survey of routing techniques for direct networks.

Many of the techniques used for routing in NoCs, such as the XY algorithm, were originally developed for computer networks and multiprocessor systems. The XY algorithm is a routing technique widely used in 2D mesh networks with wormhole switching. It works by sending packets over the network first horizontally (X dimension), then vertically (Y dimension). In the context of NoCs, XY routing proves efficient due to its simplicity of implementation and because it is deadlock-free. Works that made use of XY algorithm include HERMES network (Moraes et al., 2004) and SoCIN network (Zeferino and Susin, 2003).

Glass and Ni (1992) have proposed the so-called turn model for adaptive, livelock and deadlock-free algorithms. A turn is a change of 90° in the direction of packet transmission. The main idea of this model is to restrict the amount of turns that a packet route can go through in order to avoid the formation of cycles that cause deadlocks. Following this concept, three routing algorithms were proposed: the west-first, the north-last and negative-first. A related approach is the odd-even (OE) turn model (Chiu, 2000) for designing partially adaptive deadlock-free routing algorithms. Unlike turn model, which relies on prohibiting certain turns in order to avoid deadlock, this model restricts the locations where some types of turns can be taken. As a result, the degree of routing adaptiveness provided is more even for different source-destination pairs.

The work of Jose Duato has addressed the mathematical foundations of routing algorithms. His main interests have been in the area of adaptive routing algorithms for multicomputer networks. Most of the concepts are directly applicable to NoC. In Duato (1993), the theoretical foundation for deadlock-free adaptive routing in wormhole networks is given.
3 Ant colony optimisation

Ant algorithms, also known as ACO, are a class of heuristics search algorithms biologically inspired in the behaviour of colonies of ants, and in particular how they forage for food. One of the main ideas behind this approach is that the ants can communicate with one another through indirect means by making modifications to the concentration of highly volatile chemicals called pheromones in their neighbour environment. This form of indirect communication mediated by the environment is called stigmergy, and is typical in social insects.

The main ideas of ACO are:

1. the use of repeated simulations carried out by a population of artificial agents called ‘ants’ to generate new solutions to the problem
2. the use by the agents of stochastic local search to build the solutions in an incremental way
3. the use of information collected during past simulations (artificial pheromones) to direct future search for better solutions.

Several ant algorithms make use of the structure shown in Algorithm 1 (Dorigo et al., 2006).

Algorithm 1 ACO meta-heuristics

1: initialise parameters and pheromone trails;
2: while termination condition not met do
3: construct ant solutions;
4: local search (optional);
5: update pheromone trails;
6: end while;

In the artificial ant colony approach, each ant builds a solution by using two types of information locally accessible: problem-specific information, and information added by ants during previous iterations of the algorithm. In fact, while building a solution, each ant collects information on the problem characteristics and on its own performance, and uses this information to modify the representation of the problem, as seen locally by the other ants. The representation of the problem is modified in such a way that information contained in past good solutions can be exploited to build new and hopefully better ones.

4 ACO-based routing

The ACO, with the ability to search for paths, emerged as a powerful solution for routing problems. Thus, this paper presents the use of the ACO meta-heuristics in the construction of routing algorithms. Two models of static routing for NoCs are proposed.

4.1 Network specification

In this work, the network uses switches with five communication ports: four to communication with neighbouring switches and one for local communication with the resource. The switches are considered bufferless using no virtual channels. The network topology is a two dimension mesh. The switching technique adopted was the wormhole (Ni and McKinley, 1993). In this method, packets are divided into smaller units called flits (flow-units). It is assumed that each communication channel has a width of one flit.

4.2 Proposed routing algorithms

In this paper, we propose two algorithms to perform routing using the ACO meta-heuristics. These algorithms are called routing inspired on elitist ant system (REAS) (Dorigo et al., 1996) and routing inspired on ant colony system (RACS) (Dorigo and Gambardella, 1997). Both algorithms search for paths in an architecture characterisation graph that represents the network 2D mesh topology. These algorithms make use of multiple ant colonies, where each colony is responsible for searching the route of a packet. In this approach, each of which maintain its own pheromone and ants. Nevertheless, the colonies must exchange information in order to minimise the latency of their respective packets. Therefore, the route found by an ant in a given colony is visible to the ants in other colonies, because these packets are being transmitted simultaneously and within the same network. A simplified pseudo-code of REAS is shown in Algorithm 2.

Algorithm 2 REAS algorithm

Require: network and ACO parameters;
1: while total of cycles do
2: for $k = 1 \rightarrow \text{number of ants}$ do
3: for $g = 1 \rightarrow \text{number of packets}$ do
4: $Ant_{k,g}$ constructs a solution;
5: compute $Ant_{k,g}$ pheromone;
6: end for
7: compute the elitist pheromone;
8: accumulate the pheromone of actual ants;
9: end for
10: update the global pheromone;
11: end while
12: return best solution;

In the proposed algorithms, ants in a network node are only aware of two things: the first is the pheromone concentration in the surrounding nodes; the second is the load on a node, which dictates the waiting time in each of the four possible transmission directions.

The elitist ant system (EAS) is directly inspired by the ant system, the first of ant algorithms (Dorigo et al., 1996). The EAS is characterised mainly by the use of the concept
of elitism, in order to differentiate ants that carry out better solutions. In the REAS algorithm, ants build paths through the network selecting the next node based on (1), where \( p_{ij}^k \) is the probability of ant \( k \) going from node \( i \) to node \( j \).

\[
P_{ij}^k(t) = \begin{cases} \tau_{ij}(t)^\alpha \cdot \eta_{ij}^\beta \cdot \frac{\Delta t^k}{\Delta t} & \text{if } j \in a_k \\ 0 & \text{otherwise} \end{cases}
\]

(1)

The probability of selecting a particular direction is a function of pheromone concentration and network load in that direction. These two parameters are weighted by an importance constant \( \alpha \) and \( \beta \), respectively. The network load is used indirectly by \( \eta_{ij} \), defined in (2), where \( C_{ij} \) represents the load in transmission from \( i \) to \( j \).

\[ \eta_{ij} = 1/C_{ij} \]

(2)

At the end of each iterative cycle, the pheromone of all colonies is updated according to (3). Part of the pheromone of the previous iteration is reduced by evaporation rate \( \rho \), and then reinforced by the contribution of all \( m \) ants in the current cycle. The pheromone also receives the reinforcement of elitist ants: those that achieve the best solutions deposit their pheromone in every cycle, so directing the search in subsequent cycles.

\[
\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta \tau_k^k
\]

(3)

The second ant algorithm used in this work is described below. The RACS is very similar to REAS, with the same structure of multiple colonies being used. The algorithm on which the RACS was inspired, called ant colony system (ACS) [Dorigo and Gambardella (1997)], differs from others ant algorithms by:

\begin{itemize}
  \item the selection of next nodes in solutions building
  \item the use of a different pheromone update.
\end{itemize}

Because these two mechanisms, ACS improves over AS by increasing the importance of exploitation of information collected by previous ants with respect to exploration of the search space. The pseudo-code of RACS algorithm is shown in Algorithm 3.

Thus, the RACS uses the so-called pseudo-random proportional rule.

\[
j = \arg \max_{j \in \{1,4\}} \left\{ \tau_j \cdot \eta_j^\beta \right\} \text{ if } q \leq q_0,
\]

\[ j = S \text{ otherwise} \]

(5)

As shown in equation (5), the probability for an ant to move from node \( i \) to node \( j \) depends on a random variable \( q \), uniformly distributed over \([0, 1]\), and a parameter \( q_0 \). If \( q \leq q_0 \), then the next node is directly selected by \( \arg \max_{j \in \{1,4\}} \left\{ \tau_j \cdot \eta_j^\beta \right\} \), i.e., the direction with the largest value of \( \tau_j \cdot \eta_j^\beta \). Otherwise, the next node is defined by \( S \), that uses a selection method similar to that employed by EAS [equation (1)].

**Algorithm 3** RACS algorithm

**Require:** network parameters;

**Require:** ACS parameters;

**Require:** packets parameters;

1: \( \text{while total of cycles do} \)
2: \( \text{for } k = 1 \rightarrow \text{number of ants do} \)
3: \( \text{for } g = 1 \rightarrow \text{number of packets do} \)
4: \( \text{while node}_{\text{actual}} \neq \text{node}_{\text{destination}} \) \( \text{do} \)
5: \( \text{Ant}_k \text{\select \text{node}_{\text{actual}}} \text{;} \)
6: \( \text{calculate load of Ant}_k \text{ in node}_{\text{actual}} \text{;} \)
7: \( \text{update the local pheromone in node}_{\text{actual}} \text{;} \)
8: \( \text{node}_{\text{actual}} \leftarrow \text{node}_{\text{next}} \text{;} \)
9: \( \text{end while} \)
10: \( \text{calculate Ant}_k \text{ pheromone;} \)
11: \( \text{end for} \)
12: \( \text{if solution of ants in } k \text{ iteration is the best then} \)
13: \( \tau_{\text{best}} \leftarrow \text{pheromone of ants in } k \text{ iteration;} \)
14: \( \text{end if} \)
15: \( \text{end for} \)
16: \( \text{update the global pheromone with } \tau_{\text{best}} \text{;} \)
17: \( \text{end while} \)
18: \( \text{return best solution;} \)

The RACS algorithm also uses a double pheromone update. The offline update is applied at the end of each iteration only by the best-so-far ant.

\[
\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}^t + \rho \cdot \Delta \tau_j^k \text{ if } j \text{ belongs to best path}
\]

\[ \tau_{ij}^t \text{ otherwise} \]

(6)

The offline update is given by equation (6), where \( \Delta \tau_j \) is the reinforcement of the best ant pheromone. As said, the offline update performs a strong elitist strategy. The best ant can be the iteration-best ant, that is, the best in the current iteration, or the global-best ant, that is, the ant that made the best tour from the start of the trial.

The local update is performed by all ants in each step of construction of a solution.

\[
\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij} + \rho \cdot \tau_0
\]

(7)

This local update is defined by equation (7), where \( \rho \) is the evaporation constant, and \( \tau_0 \) is the initial pheromone at each node. In practice, ACS ants consume some of the pheromone trail on the nodes they visit. This has the effect
of decreasing the probability that a same path is used by all the ants (i.e., it favours exploration, counterbalancing this way the above-mentioned modifications that strongly favour exploitation of the collected knowledge about the problem).

In both REAS and RACS algorithms the same stop condition was used, which is a maximum number of iterative cycles.

5 Applications in NoC
In general, NoCs are developed to perform a specific application. This application can be described initially as a software that must be embedded in hardware. The EDA tool must adjust characteristics of NoC and application so that the execution is more efficient.

5.1 Task graphs
Applications can be described by a task graph, a data structure divided into blocks responsible for specific tasks. These blocks, in turn, exchange information in order to complete the application execution. The task graph is denoted by \( GT = G(T,D) \), an acyclic and weighted directed graph. Each node of \( T \) is a task, or an application processing module. In general, an operation is a well defined task, as a mathematical calculation or a data encoding. Each arc of the set \( D \) characterises the data dependencies between two tasks.

5.2 Random mapping
As mentioned in Section 1, an EDA tool can perform certain processes, such as allocation, mapping and routing of applications in NoCs. In the allocation process, cores are selected in an IP repository, and then associated with each application task. In turn, the mapping deals with how the IPs are spatially distributed in the NoC topology. Both processes are intended to optimise some characteristic of system, like execution time, silicon area and power consumption (Nedjah et al., 2012; Da Silva et al., 2010).

In this paper, the routing is emphasised. Thus, it is considered that an allocation step was previously performed, and the resource specifications are already available for use. In the mapping step, we employed a simple random mapping process. In the random mapping, each node of the application characterisation graph is associated to a node of the architecture characterisation graph. The way in which this association is made is random: a node of the application graph selects from a list of a node of the architecture graph; the chosen position is removed from the list and the process repeats until all nodes of the application graph have a defined position in the architecture graph.

The mapping also defines the number of nodes in NoC. The number \( N \) of nodes in a mesh must be sufficient to map an application with \( P \) tasks. Thus, because it is a 2D mesh, the relationship between \( N \) and \( P \) is defined by (8).

\[
N = \left[ \sqrt{P} \right]^2
\]

Figure 2 illustrates the mapping process. The tasks of the graph are associated with five nodes in a 3 × 3 network.

6 Experiments and results
A cycle-accurate network simulator was implemented in MATLAB. It supports 2D mesh networks with wormhole switching. To evaluate the performance of the proposed methods, networks were simulated with four different routing algorithms: REAS, RACS, XY and OE. The time unit adopted is the simulator cycle, where one cycle is the transmission time of one flit.

All algorithms were executed with MATLAB Version 7.7.0.471 (R008b). The simulations were performed on PCs with Intel Core i7 950 3 GHz, 8 Gb RAM and Microsoft Windows 7 Home Premium operating system.

From the information of a graph, the routing can be accomplished by identifying which packets are generated by tasks at the same level, i.e., which packets may be transmitted simultaneously. Also, a simple random mapping is employed, were each task is assigned randomly to network nodes. Algorithm 4 was used to perform this process.

Algorithm 4 Mapping and routing of application

```
Require: Task graph;
1: define size of NoC;
2: perform the mapping;
3: for \( l = 1 \rightarrow \text{#levels} \) do
4:     get all arcs in level \( l \);
5:     read \( t_{\text{start}} \) of source tasks;
6:     perform the routing;
7:     write \( t_{\text{start}} \) of destination tasks;
8: end for
9: \( t_{\text{execution}} \leftarrow t_{\text{start}}(\text{last task}) + t_{\text{comp}}(\text{last task}) \)
10: return routing paths, \( t_{\text{execution}} \);
```

6.1 Applications from E3S
The embedded systems synthesis benchmarks suite (E3S) (Dick, 2008) is a collection of task graphs, representing real applications based on embedded processors from Embedded
Microprocessor Consortium (EEMBC). It was developed to be used in system-level allocation, assignment and scheduling research. The E3S contains the characteristics of 17 embedded processors. These processors are characterised by the measured execution times of 47 different type of tasks, power consumption derived from processor data sheets, and additional information, such as die size, price, clock frequency and power consumption during idle state. In addition, E3S contains task graphs of common tasks in auto-industry, networking, telecommunication and office automation. Each one of the nodes of these task graphs is associated with a task type. A task type is a processor instruction or a set of instructions, e.g., FFT, inverse FFT, floating point operation, etc.

In this study, we used 16 graphs found in E3S, which represent serial and parallel applications. Information about the number of levels and tasks for used applications is shown in Table 1. The remaining applications are purely sequential. The AMD-ElanSC520 was selected, which is able to perform all 47 tasks.

### Table 1

<table>
<thead>
<tr>
<th>Label</th>
<th>Application name</th>
<th># tasks</th>
<th># levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>TG1</td>
<td>auto-indust-tg0</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>TG2</td>
<td>auto-indust-tg1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>TG3</td>
<td>auto-indust-tg2</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TG4</td>
<td>auto-indust-tg3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>TG5</td>
<td>consumer-tg0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>TG6</td>
<td>consumer-tg1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>TG7</td>
<td>networking-tg1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>TG8</td>
<td>networking-tg2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>TG9</td>
<td>networking-tg3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>TG10</td>
<td>office-tg0</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>TG11</td>
<td>telecom-tg0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>TG12</td>
<td>telecom-tg1</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>TG13</td>
<td>telecom-tg2</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>TG14</td>
<td>telecom-tg3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>TG15</td>
<td>telecom-tg4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>TG16</td>
<td>telecom-tg5</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

### 6.2 SegImag application

Another application used in this work is the segmentation of images for object recognition, SegImag (Marcon, 2005). This application aims to accelerate the process of identifying the number of objects in an image. For this purpose, the original image must be split in parts, where each segment is handled by an auxiliary processor. In addition, the SegImag contains two other processing elements: a central processor, which receives the results of each auxiliary processor, and an external memory which stores the image to be segmented.

In the original implementation of SegImag, the amount of auxiliary processors is parameterised. The segmentation is directly related to the amount of processing elements, since each segment must be processed by an auxiliary processor.

In the present study, we used the implementation shown in Da Silva et al. (2009a), which the image is segmented into four parts. The task graph of this application is shown in Figure 3. The values in each arc represent the amount of bits transmitted between tasks, considering an image with 640 × 480 pixels. The run-time values were based on allocation results presented in Da Silva et al. (2009a), where each task in SegImag task graph was associated with a task performed by a processor repository E3S. This information concerning the task data is organised in Table 2.

### Table 2

<table>
<thead>
<tr>
<th>Label</th>
<th>Task name</th>
<th>id</th>
<th>Task time (cycles)</th>
<th>Proc. name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td>Decompress JPEG</td>
<td>455</td>
<td>$7 \times 10^7$</td>
<td>ST20C2</td>
</tr>
<tr>
<td>PI</td>
<td>Fixed point complex FFT – Data3 (sine)</td>
<td>449</td>
<td>$1.2 \times 10^5$</td>
<td>ST20C2</td>
</tr>
<tr>
<td>PF</td>
<td>Basic floating point</td>
<td>371</td>
<td>$9 \times 10^3$</td>
<td>MPC555</td>
</tr>
<tr>
<td>PCV</td>
<td>Autocorrelation – Data2 (sine)</td>
<td>439</td>
<td>$6.9 \times 10^4$</td>
<td>ST20C2</td>
</tr>
<tr>
<td>MI</td>
<td>Compress JPEG</td>
<td>454</td>
<td>$8.7 \times 10^7$</td>
<td>ST20C2</td>
</tr>
</tbody>
</table>

### Figure 3

Task graph of SegImag application

### 6.3 Results

The simulations were performed by submitting applications to four different routing algorithms and measuring its total execution time. This consists of execution of all individual tasks on a critical path plus the communication time of these tasks. The so-called packet delay is the difference between the value obtained using a specific routing algorithm and the optimal value of the network without congestion. To
calculate this ideal value, we used a modified XY algorithm, called dummy XY. In this routing, the XY algorithm is used to define the communication time using shortest paths. But unlike the real XY (and any other routing algorithm), the potential congestion delays are not counted.

Figure 4 shown results of simulations with real world applications. The values of packet delay are presented for the four routing algorithms – XY, OE, REAS and RACS. These values are a mean of packet delay in ten different mappings. Results of only five applications are shown. To other 11 applications, all the four routing algorithms were able to find the best path, i.e., no delay congestion occurred. In four of the five applications, ACO-based routing found better results than other algorithms.

7 Conclusions

Static routing is an efficient solution in NoCs designed to run always the same set of applications, since communication paths need be defined only one time. In this paper, we propose the use of ACO-based algorithms in the optimisation of paths in the static routing step in NoC design. The performance of these algorithms was evaluated using task graphs of real world applications, such SegImage and those found in E3S Suite. Best results were obtained with REAS algorithm. Future work may study how to increase the performance of proposed algorithms and the use of other ant algorithms in the routing task.

Acknowledgements

We are grateful to Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Coordenação de Aperfeiçoamento de Pessoal de Ensino Superior (CAPES) for their continuous financial support.

References


