PhoNoCMap: an Application Mapping Tool for Photonic Networks-on-Chip User Manual

Edoardo Fusella

Department of Electrical Engineering and Information Technologies,
University of Naples Federico II, Italy

Email: edoardo.fusella@unina.it

January 1, 2016

1 Introduction

This document is used to completely describe PhoNoCMap to a person wishing to use or modify it.

1.1 Background

The recent grow in transistor integration has allowed a shift from the single-core era to the multicore paradigm during the last decades. In a large-scale multicore scenario, an energy-efficient on-chip communication fabric is the key ingredient ensuring performance scalability. While traditional electronic interconnects are constrained by physical limitations in terms of power dissipation, latency, and bandwidth, silicon Photonics appears a promising path to energy-efficient ultra-high bandwidth on-chip communication. In order to exploit the potential advantages of photonics, we need to deal with specific electromagnetic effects, like insertion loss and the crosstalk noise, that potentially impact the optical NoC architecture to a large extent.

Electromagnetic effects should be a major goal when designing a photonic NoC architecture, since a high power loss or crosstalk noise may easily result in a network leading to poor performance or even in an inoperable architecture. The NoC architecture should be customized for a target application, when its traffic characteristics are known at design time, which is the case

for most embedded applications running on multiprocessor Systems-on-Chip (MPSoCs). The problem of mapping a set of given IP cores to the NoC tiles is illustrated in Figure 1. An application, previously modeled as a graph of concurrent tasks, is assigned to a set of available cores by mapping different functions to different regions of the system, so that the traffic exhibits patterns optimized for the electromagnetic effects.

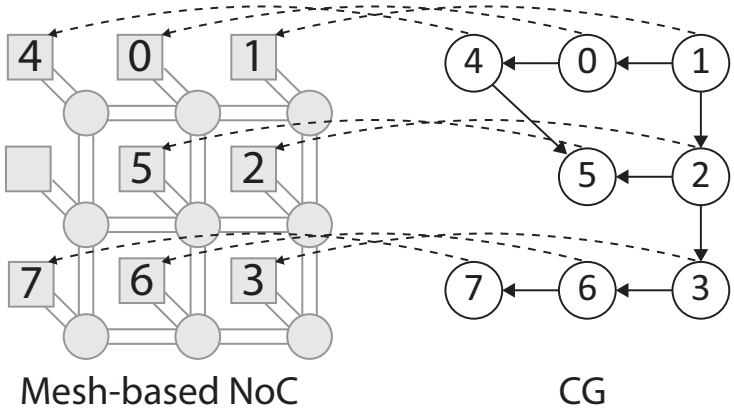

Figure 1: A mesh-based on-chip architecture and an example of mapping problem.

1.2 What is PhoNoCMap?

PhoNoCMap (photonic network-on-chip application mapping tool) was originally designed to allow us to investigate application-specific silicon photonic NoCs taking into account the photonic distinctive electro-magnetic effects that potentially impact on the NoC architecture constraining the network scalability. We ended up with a tool that is suited to automatically map the IP cores onto a photonic NoC architecture such that the metrics of interest are optimized. It helps system architects to explore how mapping solutions impact the performance/costs of a particular computing system and find the best mapping solution for a certain application. The tool contains built-in analytical models for estimating both power loss and crosstalk noise, thereby allowing accurate estimates, and is fully customizable since new topologies, routing algorithms, optical router architectures, and mapping optimization strategies can be easily added.

1.3 How to get PhoNoCMap

The following are step-by-step instructions for setting up PhoenixSim on your local machine to run mapping optimizations, or modify the code. Note that PhoNoCMap requires Java 7.
Getting PhoNoCMap:

1. PhoNoCMap requires Java 7. Please ensure you have the right java version installed on your PC.
2. Download the PhoNoCMap zip file from:
   `http://wpage.unina.it/edoardo.fusella/phonocmap/`
3. Extract it somewhere on your computer.
4. Now you are ready to enjoy PhoNoCMap.

Running an application mapping optimization:

1. Open the *app* folder and create a new application communication file describing the communication requirement of a target application (see section 2.3).
2. Lunch a Windows Command Prompt or a Unix shell.
3. Browse to the PhoNoCMap directory.
4. Lunch PhoNoCMap (java -jar PhoNoCMap.jar).
5. Configure all the PhoNoCMap parameters moving among the different windows.
6. In the *Execution* window, click the Start Mapping Optimization button.
7. View the results in the white text area and eventually in the result file located in the *output* folder (see section 2.6).

1.4 Required reading

Before using this tool, you should be familiar with some things. For a comprehensive overview of circuit-switched photonic networks, you can read [1]. For power loss and crosstalk analysis of photonic NoCs, you can read [2, 3, 4]. Our paper [5] briefly presents PhoNoCMap, while the works presented in [6, 7] rely on PhoNoCMap to perform the experimental evaluation.

1.5 Copyright And License

PhoNoCMap is made openly available under the following license. Please cite the following paper if PhoNoCMap is used for your research:
E. Fusella and A. Cilardo, PhoNoCMap: an application mapping tool for photonic networks-on-chip, in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2016.* IEEE, 2016.

1.6 The rest of this document

Chapter 2 explains how the simulator works and gives some details. Chapter 3 briefly describes the models that are used for estimating both power loss and crosstalk noise. Chapter 4 describes the ways in which the simulator could be extended to support additional custom components and functionality to suit the needs of a particular NoC architecture.

2 PhoNoCMap

This chapter explains the overall structure of PhoNoCMap, and how to use PhoNoCMap.

2.1 Folder hierarchy

The components you will find in the PhoNoCMap folder are organized into various subfolders depending on their functions. In the following, we will enumerate and describe the contents of these subfolders:

- app - contains the application xml files used to describe the communication requirements of an application through a Communication Graph (CG);
- bin - contains the tool compiled Java files;
- doc - contains the API documentation in HTML format from Java source code generated using Javadoc;
- lib - contains all the jar files belonging to external libraries;
- output - contains the files where are stored the results of the mapping optimizations;
- src - contains the tool source Java files.

In addition, the PhoNoCMap folder contains:

- PhoNoCMap.jar - the PhoNoCMap executable jar file;
- default_parameters.xml - the xml file containing the default settings of the GUI. It is required to edit this file only if you plan to customize PhoNoCMap. See Chapter 4 if you are interested in that;
- userGuide.pdf - the PhoNoCMap user guide that you are reading.

2.2 Architecture description

The NoC architecture is described by the topology, the routing algorithm and the optical router microarchitecture. Figure 2 shows a screenshot of the NoC architecture window. While PhoNoCMap was initially planned and developed targeting photonic NoC architectures based on direct topologies (tori) with dimension order routing, both the underlying models and the tool architecture can be easy extended to any other photonic architecture. See Chapter 4 for more details.

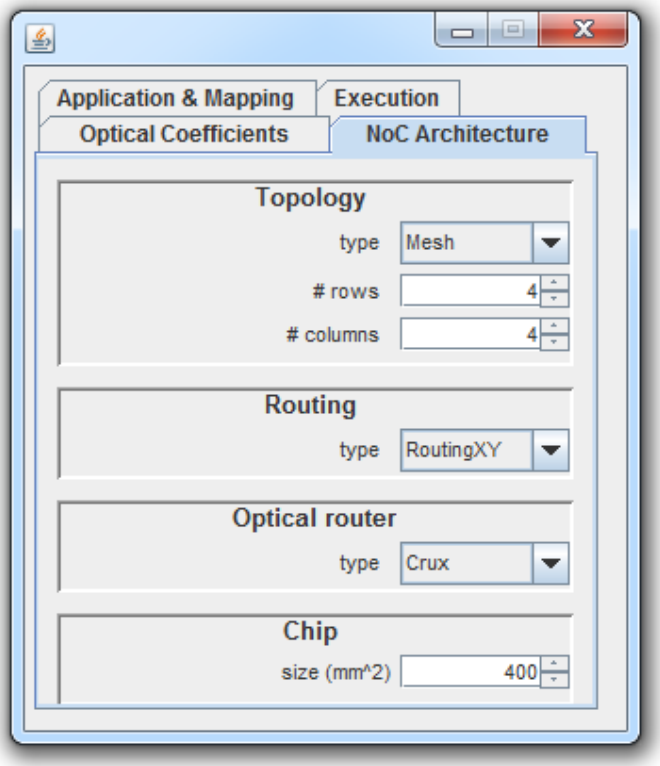

Figure 2: A screenshot of the *NoC Architecture* window.

2.3 Application description

PhoNoCMap supports Communication Graphs (CGs) for describing the application communication requirements:

Definition 1 *A* Communication Graph $CG = G(C, E)$ *is a directed graph where each vertex* $c_i \in C$ *is an IP core and* $e_{i,j} \in E$ *is the edge between cores* c_i *and* c_j *characterizing the communication between them. Each edge* $e_{i,j}$ *can optionally have several attributes expressing application-specific information:*

- $b(e_{i,j})$ *the communication bandwidth requirement from core* c_i *to* c_j*, i.e. the minimum bandwidth (Mb/s) that should be provided by the communication architecture.*

The Communication Graph is stored in an application xml file as Figure 3 shows. For each application, it is required to specify the total number of IP cores and every edge in the CG. The communication bandwidth requirement

can be optionally specified. This information is used for a more accurate laser power consumption estimation (See Section 3 for more details).

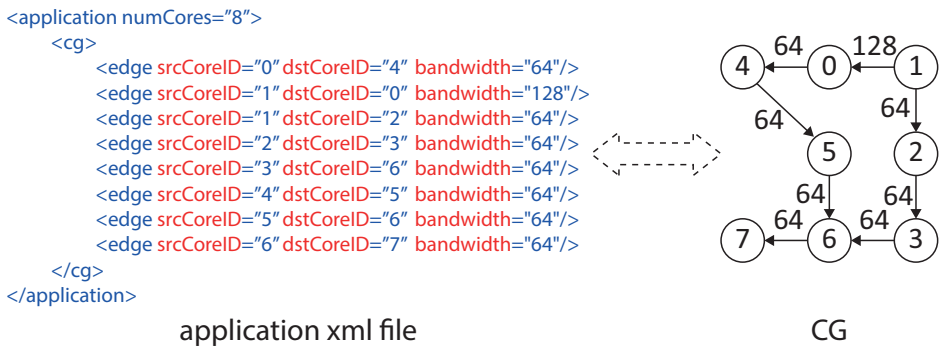

Figure 3: An application xml file and the related Communication Graph.

Currently, PhoNoCMap is released with eight real typical streaming video and image processing applications, namely:

- *263dec mp3dec*, which is a H.263 video decoder and MP3 audio decoder (mapped onto 14 cores);
- *263enc mp3enc*, which is a H.263 video encoder and MP3 audio encoder (mapped onto 12 cores);
- *DVOPD*, which is a dual video object plane decoder (mapped onto 32 cores);
- *MPEG-4*, which is a MPEG4 decoder (mapped onto 12 cores);
- *MWD*, which is a multi-window display (mapped onto 12 cores);
- *PIP*, which is a picture-in-picture application (mapped onto 8 cores);
- *VOPD*, which is a video object plane decoder (mapped onto 16 cores);
- *Wavelet*, which is a wavelet transform application (mapped onto 22 cores).

Note that the application file name must be specified, in the relative GUI field, without the *.xml* extension.

2.4 Mapping strategy

The mapping strategy is described by the optimization algorithm, the optimization objective, the number of iterations and the stop condition. In case of genetic algorithm, it is possible to specify the population and offspring sizes. PhoNoCMap is released with three different algorithms:

- a random search
- a genetic algorithm
- a list algorithm

The metrics driving the optimization phase are three:

- the worst-case crosstalk noise
- the worst-case insertion loss
- the average laser power consumption

As regards the stop condition, it is possible to choose between two possibilities. The default stop criterion (condition 0) is based on the total number of iterations. Differently, it is possible to choose a stop criterion based on distance convergence (condition 1). Basically, the optimization is stopped when there isn't any appreciable improvement in the consecutive phenotypes of the new populations that are being created for a certain number of iterations. The main advantage is that it is not necessary to determine the exact number of iterations required to reach a satisfactory solution.

Figure 4 shows a screenshot of the Mapping strategy window.

2.5 Optical Coefficients

Figure 5 illustrates how optical signals and crosstalk noise propagate for both the parallel PSE (PPSE) and the crossing PSE (CPSE) in ON or OFF resonance and in case of waveguide crossing. The crosstalk noise arises when two optical signals reach simultaneously a waveguide crossing or a PSE. PSEs (Figure 5 a-d) are made up of a microring resonator and two waveguides. When an optical signal injected into the input port matches the wavelength of the microring resonator resonance frequency, then it is coupled into the ring and steered to the drop port (Figure 5(b) and (d)). Otherwise, the signal propagates to the through port (Figure 5(a) and (c)). The output power at each port of both the PPSE and CPSE in both the ON and OFF resonance state is evaluated as a function of the input power and the loss/crosstalk

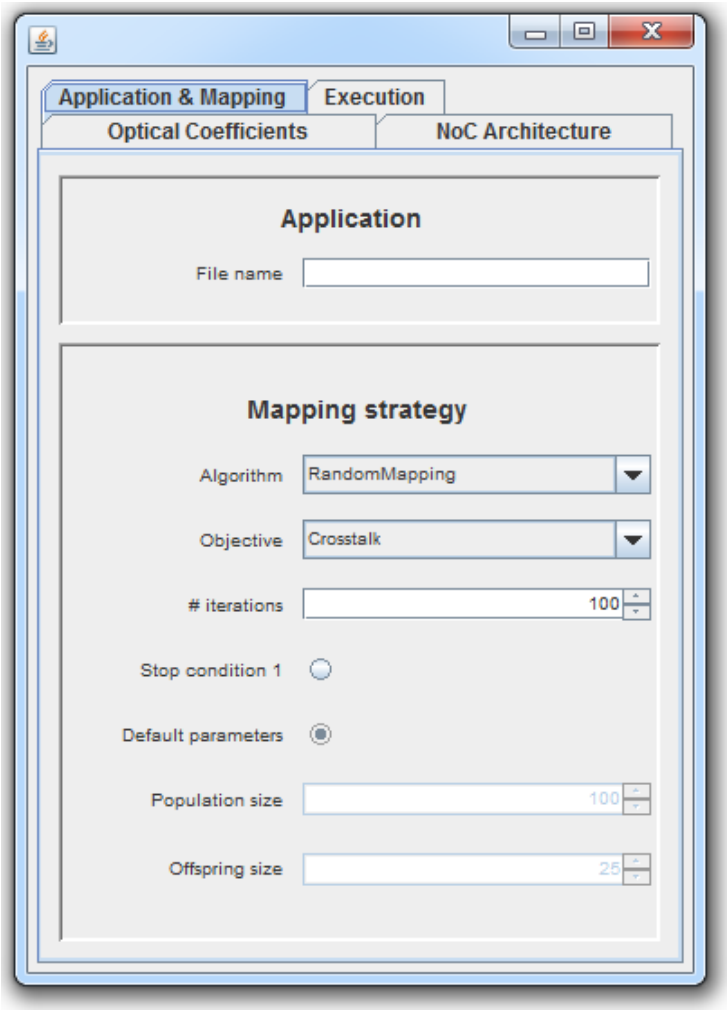

Figure 4: A screenshot of the *Application and Mapping* window.

coefficients presented in this section. Table 1 shows the default values derived from the scientific literature, while the models covered by our methodology are explained in the sections 3. All the optical parameters are configurable in the *Optical Coefficients* window. In addition, it is possible to specify the photodetector sensitivity, the laser efficiency and the modulation rate. These values are used for the laser power consumption estimation as explained in Section 3.

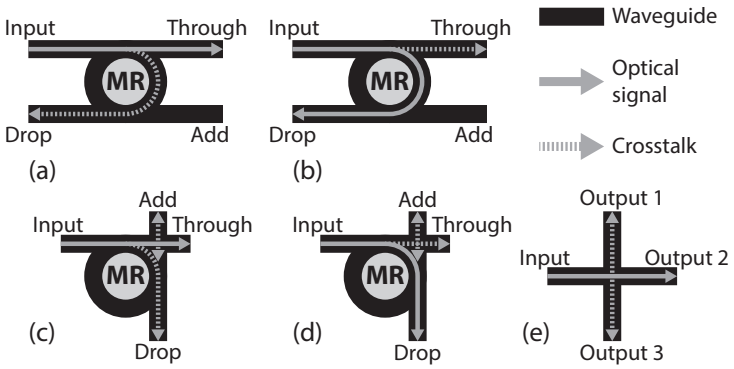

Figure 5: How optical signal and crosstalk noise propagate through: (a) Parallel PSE in OFF state; (b) Parallel PSE in ON state; (c) Crossing PSE in OFF state; (d) Crossing PSE in ON state; (e) Waveguide Crossing.

Table 1: Loss and crosstalk parameters

Parameter	**Notation**	**Value**	**Ref.**
Crossing loss	L_c	0.04 dB	[8]
Propagation Loss in Silicon	L_p	0.274 dB/cm	[9]
Power loss per PPSE in OFF state	$L_{p,off}$	0.005 dB	[10]
Power loss per PPSE in ON state	$L_{p,on}$	0.5 dB	[10]
Power loss per CPSE in OFF state	$L_{c,off}$	0.045 dB	
Power loss per CPSE in ON state	$L_{c,on}$	0.5 dB	[11]
Crossing's crosstalk coefficient	K_c	40 dB	[8]
Crosstalk coefficient per PSE in OFF state	$K_{p,off}$	20 dB	[10]
Crosstalk coefficient per PSE in ON state	$K_{p,on}$	25 dB	[10]
Photodetector sensitivity	$S_{detector}$	-14.2 dBm	[12]
Laser efficiency	E_{laser}	30 %	[13]
Modulation rate	Mod	10 Gb/s	[10]

2.6 Outputs in PhoNoCMap

PhoNoCMap supports two different output formats. First, at the end of a mapping optimization run, a brief view of the results is shown in the white text area located in the *Execution* window. Fig 6 shows an example. The mapping solution is represented as a $n \times m$ matrix, where each entry, that is in one-to-one correspondence with a tile of the NoC, contains an identifier of the core mapped to that tile. Then, the worst case path leading to the worst case signal-to-noise ratio (SNR) or power loss is showed. In case of a crosstalk optimization, the different optical communications that contribute to the worst case crosstalk noise are highlighted. Finally, in case of a crosstalk optimization, the worst case signal and noise attenuations as well as the worst case SNR are showed, while, in case if a power loss optimization, just the

signal attenuation is shown.

In addition, a radio button, located in the *Execution* window, enables the writing of an output file in the output folder. The output file name depends on the application, the NoC architecture and the mapping algorithm in the following way: *application name _ topology _ NoC size _ routing algorithm _ optical router _ mapping algorithm _ mapping objective*. This file contains all the required information about the tool setup as well as the mapping optimization results.

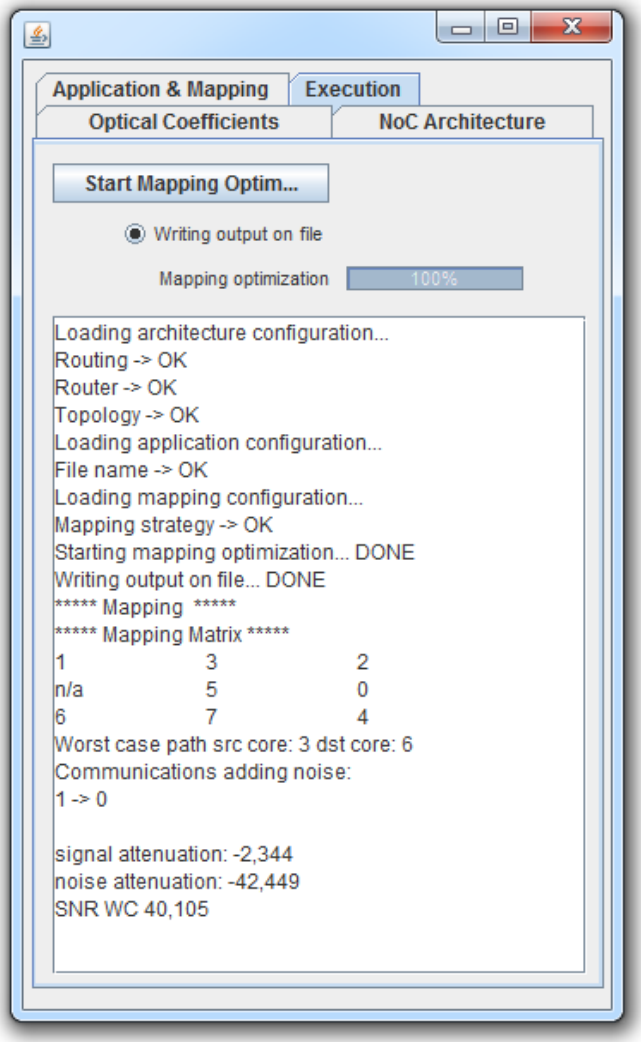

Figure 6: A screenshot of the *Execution* window at the end of an optimization run.

3 Modeling

The nodes are organized in a two-dimensional topology[1] where the connectivity is achieved using exclusive links between adjacent nodes. The architecture is composed of $n \times m$ tiles, each containing an optical router connected with an IP core and with the four neighboring tiles. Each connection is made up of two unidirectional silicon waveguides connecting two routers or a router and an IP core.

3.1 The Optical Models

We adapted the model presented in [3] introducing the following modifications:

- The crosstalk noise on the add port as well as the light that reflects back on the input port are neglected.
- We consider only the first-order crosstalk noise and hence $K_i K_j = 0$, with $K_i, K_j \in \{K_c, K_{p,off}, K_{p,on}\}$
- We neglected the power loss that affects the crosstalk noise inside the switch where the crosstalk noise is generated. As a consequence, $K_i L_i = K_i$, with $K_i \in \{K_c, K_{p,off}, K_{p,on}\}$ and $L_i \in \{L_c, L_p, L_{p,off}, L_{p,on}, L_{c,off}, L_{c,on}\}$

Based on the above considerations, the equations presented in [3] are semplified as follows.

$$P_{Tppse,off} = L_{p,off} P_{in} \tag{1a}$$
$$P_{Dppse,off} = K_{p,off} P_{in} \tag{1b}$$
$$P_{Dppse,on} = L_{p,on} P_{in} \tag{1c}$$
$$P_{Tppse,on} = K_{p,on} P_{in} \tag{1d}$$
$$P_{Tcpse,off} = L_{c,off} P_{in} \tag{1e}$$
$$P_{Dcpse,off} = (K_{p,off} + K_c)\, P_{in} \tag{1f}$$
$$P_{Dcpse,on} = L_{c,on} P_{in} \tag{1g}$$
$$P_{Tcpse,on} = K_{p,on} P_{in} \tag{1h}$$
$$P_{out2} = L_c P_{in} \tag{1i}$$
$$P_{out1} = P_{out3} = K_c P_{in} \tag{1j}$$

[1] Notice that, even if we rely on direct topologies, such as torus networks, the proposed approach could be simply extended to any other topology (See Chapter 4 for more details).

Equations (1a) and (1b) and equations (1d) and (1c) give the output powers at the through and drop ports for the PPSE respectively in the OFF and ON state, while equations (1e) and (1f) and equations (1h) and (1g) are the same equations for the CPSE. Finally, equations (1i) and (1j) give the power detected at the output port of a waveguide crossing.

Based on the above model, it is possible to evaluate the power loss ($L^{sw}_{in,out}$) that a signal is subject to when traveling from port *in* to port *out* of an optical router and the crosstalk noise $K^{sw}_{in,out,i,j}$ that a signal is subject to when traveling from port *in* to port *out* due to another optical signal traveling from port i to port j.

$L^{sw}_{in,out}$ and $K^{sw}_{in,out,i,j}$ are used to evaluate the power loss and the crosstalk noise at the network level respectively taking into account all the losses and all the crosstalk noises in each hop along a path between a source and a destination.

Finally, note that the insertion loss impacts significantly on the laser power consumption: it is possible to generate at the laser source just enough power to ensure a proper detection of the optical signal at the photodetectors. As a consequence, the power required to generate a new optical signal depends the power loss the signal is subject to when traveling to the destination. By reducing the average insertion loss, it is possible to reduce the whole laser power consumption. In that respect, the laser power consumption P_{laser} is evaluated as

$$P_{laser} = \frac{S_{detector} L_{src,dst}}{E_{laser}} \tag{2}$$

where $S_{detector}$ is the photodetector sensitivity, $0 \leq E_{laser} \leq 1$ is the laser efficiency and $L_{src,dst}$ is the power loss along the path.

4 How to ...

PhoNoCMap requires Java 7. If you will make some changes to the source code, you will need the JDK installed in order to re-compile PhoNoCMap. PhoNoCMap is fully customizable since new topologies, routing algorithms, optical router architectures, and mapping optimization strategies can be simply added. If you browse in the *src* folder hierarchy you will see different packages:

- building_blocks - contains all the basic building blocks of a photonic NoCs, i.e. the microring resonators, the waveguides and the waveguide crossings;

- genetic_mapping - contains all the classes that handle the genetic mapping optimization;
- gui - contains the graphical user interface class;
- main - contains the core of the PhoNoCMap tool;
- mapping_strategies - contains the mapping optimization algorithms;
- routers - contains the optical routers;
- routing - contains the routing algorithms;
- topologies - contains the topologies.

For each of the above customizable features, there is a package containing all the classes implementing that particular feature. A new class can be simply created through the inheritance mechanism. Once the new class is added to the right folder, it is required to modify the *default_parameters.xml* file located in the PhoNoCMap folder. This file is parsed in order to update the feature lists in the GUI at every run. Once the xml file is modified, you will see the new feature in the GUI. Note that the name of the feature inside the xml file must be the class name of the feature. As an example if you want to create a new optical router called routerX, you should create a new class named RouterX that extend the abstract class main.Router. The RouterX class file must be placed in the routers package and in the *default_parameters.xml* file you must specify RouterX under the right class_name attribute.

4.1 Create my own optical router

In a tiled photonic NoC, each tile is made up of an IP core coupled with an optical router. Optical routers are characterized by a proper layout made up of several waveguides and microring resonators. In addition, due to the planar nature of physical layouts achievable on silicon, a certain number of waveguide crossings are unavoidable. This makes the optical router a critical component impacting on both the crosstalk noise and power loss of the whole network. For this reason, researchers have proposed several switch architectures. PhoNoCMap is released with six optical routers:

- OXY [14] (Fig. 7(a))
- ODOR [15] (Fig. 7(b))
- Cygnus [16] (Fig. 7(c))

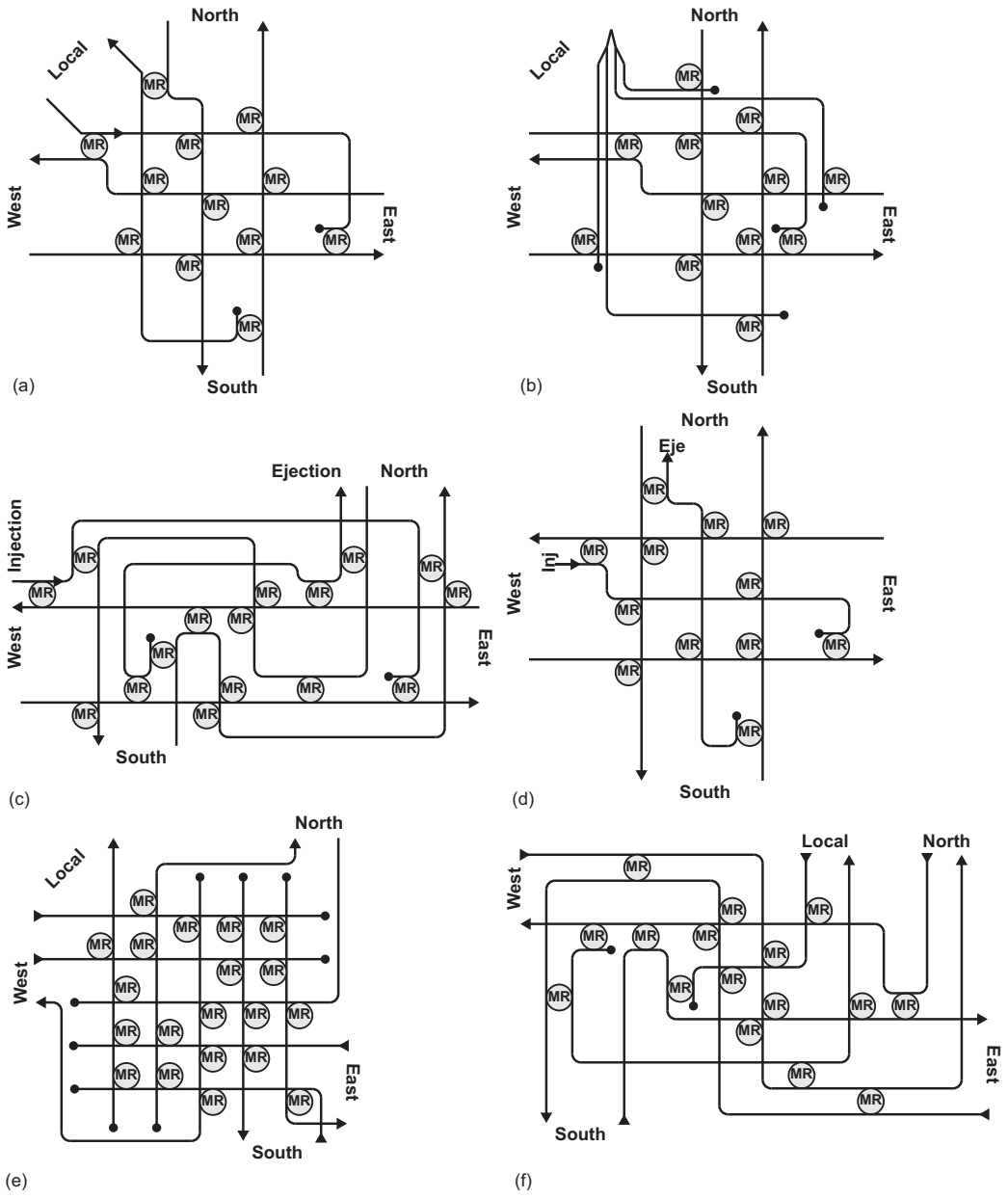

Figure 7: The switches considered in PhoNoCMap: (a) OXY [14]; (b) ODOR [15]; (c) Cygnus [16]; (d) Crux [17]; (e) Crossbar [18]; (f) Optical router [19].

- Crux [17] (Fig. 7(d))
- Crossbar [18] (Fig. 7(e))

- Optical router [19] (Fig. 7(f))

You can design your own router by extending the *Router* class located in the package *main* and overriding the *initialize()* method. This method must set the required information about the router design, i.e. the waveguides, the waveguide crossings and the microring resonators. In that respect, the Router class provides three HashMap objects:

- *HashMap <Integer, MicroringResonator> rings*: it stores all the MicroringResonator objects required in the optical router.
- *HashMap <Integer, Crossing> crossings*: it stores all the Crossing objects required in the optical router.
- *HashMap <Integer, Waveguide> waveguides*: it stores all the Waveguide objects required in the optical router.

In addition it is required to call the parent constructor of your router class specifying the following arguments:

1. number of ports of your router
2. number of microring resonators
3. number of the waveguide crossings
4. number of the waveguides

In the following, we will give a brief overview of the main building blocks.

4.1.1 Waveguide

Waveguides, the photonic counterpart of electronic wires, are the main elements of photonic NoCs and are used at the router and network levels. A Waveguide object can connect two tiles, two waveguide crossings or a tile with a waveguide crossing. Fig. 8 shows three possible ways to implement the connectivity.

When you create a new Waveguide, the constructor takes as input the following parameters:

1. the Waveguide ID
2. the input waveguide crossing ID
3. the output waveguide crossing ID

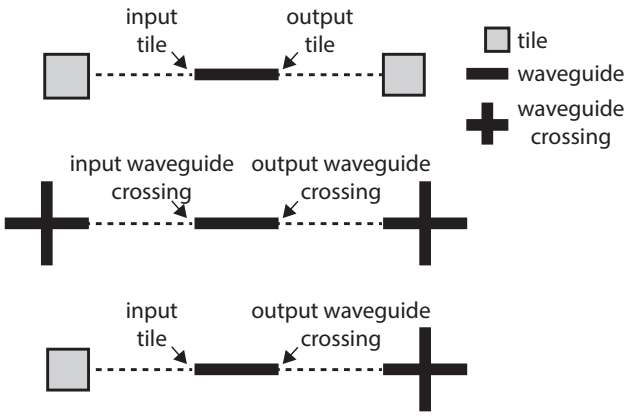

Figure 8: Example of a Waveguide object.

4.1.2 Waveguide Crossing

Due to the planar nature of 2D topologies, waveguide crossings are easily obtained. The Crossing objects are used to specify the occurrences of these waveguide crossings. As shown in Fig. 9, there are two input waveguides and two output waveguides. The Crossing constructor takes as input the following parameters:

1. the ID of the Crossing object
2. the ID of the Waveguide connected to the input 0
3. the ID of the Waveguide connected to the input 1
4. the ID of the Waveguide connected to the output 0
5. the ID of the Waveguide connected to the output 1

In addition, the Crossing object is used to implement the crossing PSE. In this case, one or two microring resonators could be added to provide the switching facilities. Their IDs should be specified in the corresponding fields using the setting methods, otherwise the value is set to -1.

4.1.3 MicroringResonator

The microring resonator is the basic building block required for providing the switching facilities. A MicroringResonator object can be used in a crossing PSE or a parallel PSE, and is placed between an input and an output waveguide (Figure 5). In order to easily implement routing facilities, each MicroringResonator object must specify which output port of the optical

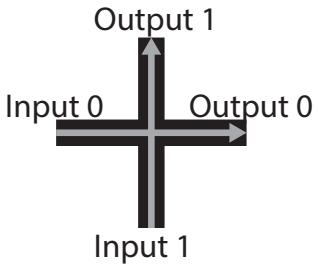

Figure 9: Inputs and outputs of a Crossing object.

router a signal takes when the ring is placed in a ON resonance state. The constructor takes:

1. the ID of the microring resonator
2. the type of the microring resonator: 0 → PPSE, 1 → CPSE
3. the ID of the input Waveguide
4. the ID of the output Waveguide reached in case of ON resonance state
5. the output port a signal takes when the ring is placed in a ON resonance state

4.2 Create my own topology

PhoNoCMap is released with four topologies:

- Mesh (Fig. 10(a))
- Unfolded torus (Fig. 10(b))
- Folded torus (Fig. 10(c))
- Unfolded torus with an optimized floorplan [2] exhibiting a reduced number of waveguide crossings (Fig. 10(d))

In order to create a new topology it is possible to extend the class *Topology* in the package *main* and override the *calcTile_list* method. This method is responsible for creating the required connections between the tiles of the topology. In this regards, it is possible to exploit the various building blocks, located in the *building_blocks* package, or extend the library with other novel building blocks. The *calcTile_list* method accepts three input parameters:

1. *HashMap <Integer, Tile> tiles*: it stores the Tile objects.

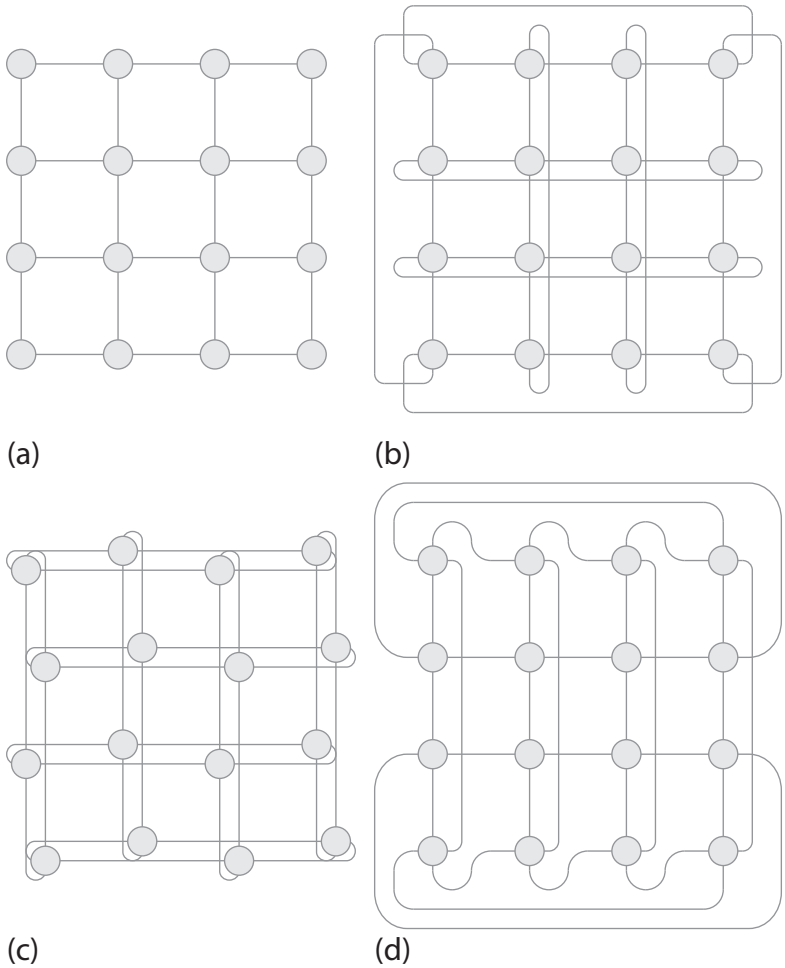

Figure 10: The topologies considered in PhoNoCMap: (a) Mesh; (b) Unfolded torus; (c) Folded torus; (d) Unfolded torus with an optimized floorplan [2].

2. *HashMap <Integer, Waveguide> waveguides*: it stores the Waveguide objects.

3. *HashMap <Integer, Crossing> crossings*: it stores the Waveguide Crossing objects.

4.3 Create my own routing algorithm

When a message should be transmitted between two tiles, it is necessary to decide which route to take. The routing algorithm defines the strategy used

to route packets from a source to a destination tile. PhoNoCMap is released with XY dimension-order routing algorithms for each of the four topologies:

- XY routing for mesh
- XY routing for unfolded torus
- XY routing for folded torus
- XY routing for unfolded torus with the optimized floorplan

In order to implement a new routing algorithm you must extend the *Routing* class present in the package *main* and override the *calcOutputPort* and *calcInputPort* methods.

The first method, *calcOutputPort*, is used to calculate the output port of a source tile that an optical signal should take in order to reach a destination tile. The second method, *calcInputPort*, is used to calculate the input port of the destination tile that the optical signal takes when coming from a source tile.

4.4 Create my own mapping optimization algorithm

PhoNoCMap is released with three algorithms:

- a random search
- a genetic algorithm
- a list algorithm

In order to create your own mapping algorithm you must implement the interface MappingStrategyInterface in the package main. This interface requires that the *initialize* and *map* methods are implemented.

The first method must set the mapping solution evaluator and the GUI progress bar. In this respect, the method takes as input an instance of the MappingEvaluator and JProgressBar objects. The first is required to evaluate the mapping solutions, while the second is used to estimate and update the algorithm execution progress.

The second method handles the mapping optimization phase and takes as input the *Application* and the *NoCarchitecture* objects. This method should contain your custom mapping strategy algorithm. The method must return a MappingSolution object that contains the solution found. The *MappingSolution object* contains a $M \times N$ matrix where each entry, that is in one-to-one correspondence with a tile of the NoC, contains an identifier of the core mapped to that tile.

References

[1] K. Bergman, L. P. Carloni, A. Biberman, J. Chan, and G. Hendry, *Photonic Network-on-Chip Design*. Springer, 2013.

[2] K. Feng, Y. Ye, and J. Xu, "A formal study on topology and floorplan characteristics of mesh and torus-based optical networks-on-chip," *Microprocessors and Microsystems*, vol. 37, no. 8, pp. 941–952, 2013.

[3] Y. Xie, M. Nikdast, J. Xu, X. Wu, W. Zhang, Y. Ye, X. Wang, Z. Wang, and W. Liu, "Formal worst-case analysis of crosstalk noise in mesh-based optical networks-on-chip," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 10, pp. 1823–1836, 2013.

[4] M. Nikdast, J. Xu, X. Wu, W. Zhang, Y. Ye, X. Wang, Z. Wang, and Z. Wang, "Systematic analysis of crosstalk noise in folded-torus-based optical networks-on-chip." *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 33, no. 3, pp. 437–450, 2014.

[5] E. Fusella and A. Cilardo, "PhoNoCMap: an application mapping tool for photonic networks-on-chip," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2016*. IEEE, 2016.

[6] E. Fusella and A. Cilardo, "Crosstalk-aware mapping for tile-based optical network-on-chip," in *High Performance Computing and Communications, 2015 IEEE 7th Intl Symp on Cyberspace Safety and Security, 2015 IEEE 12th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), 2015 IEEE Intl Conf on*. IEEE, 2015.

[7] E. Fusella and A. Cilardo, "Minimizing power loss in optical networks-on-chip through application-specific mapping," *Microprocessors and Microsystems*, 2016.

[8] W. Ding, D. Tang, Y. Liu, L. Chen, and X. Sun, "Compact and low crosstalk waveguide crossing using impedance matched metamaterial," *Applied Physics Letters*, vol. 96, no. 11, p. 111114, 2010.

[9] P. Dong, W. Qian, S. Liao, H. Liang, C.-C. Kung, N.-N. Feng, R. Shafiiha, J. Fong, D. Feng, A. V. Krishnamoorthy *et al.*, "Low loss silicon waveguides for application of optical interconnects," in *Proc. IEEE Photon. Soc. Summer Topical Meeting Ser*, 2010, pp. 191–192.

[10] J. Chan, G. Hendry, K. Bergman, and L. P. Carloni, "Physical-layer modeling and system-level design of chip-scale photonic interconnection

networks," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 10, pp. 1507–1520, 2011.

[11] B. G. Lee, A. Biberman, P. Dong, M. Lipson, and K. Bergman, "All-optical comb switch for multiwavelength message routing in silicon photonic networks," *Photonics Technology Letters, IEEE*, vol. 20, no. 10, pp. 767–769, 2008.

[12] G. Masini, G. Capellini, J. Witzens, and C. Gunn, "A four-channel, 10 gbps monolithic optical receiver in 130nm cmos with integrated ge waveguide photodetectors," in *National Fiber Optic Engineers Conference*. Optical Society of America, 2007, p. PDP31.

[13] J. Ahn, M. Fiorentino, R. G. Beausoleil, N. Binkert, A. Davis, D. Fattal, N. P. Jouppi, M. McLaren, C. M. Santori, R. S. Schreiber *et al.*, "Devices and architectures for photonic chip-scale integration," *Applied Physics A*, vol. 95, no. 4, pp. 989–997, 2009.

[14] H. Gu, J. Xu, and Z. Wang, "A novel optical mesh network-on-chip for gigascale systems-on-chip," in *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*. IEEE, 2008, pp. 1728–1731.

[15] H. Gu, J. Xu, and Z. Wang, "Odor: a microresonator-based high-performance low-cost router for optical networks-on-chip," in *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*. ACM, 2008, pp. 203–208.

[16] H. Gu, K. H. Mo, J. Xu, and W. Zhang, "A low-power low-cost optical router for optical networks-on-chip in multiprocessor systems-on-chip," in *VLSI, 2009. ISVLSI'09. IEEE Computer Society Annual Symposium on*. IEEE, 2009, pp. 19–24.

[17] Y. Xie, M. Nikdast, J. Xu, W. Zhang, Q. Li, X. Wu, Y. Ye, X. Wang, and W. Liu, "Crosstalk noise and bit error rate analysis for optical network-on-chip," in *Proceedings of the 47th Design Automation Conference*. ACM, 2010, pp. 657–660.

[18] A. W. Poon, X. Luo, F. Xu, and H. Chen, "Cascaded microresonator-based matrix switch for silicon on-chip optical interconnection," *Proceedings of the IEEE*, vol. 97, no. 7, pp. 1216–1238, 2009.

[19] R. Ji, L. Yang, L. Zhang, Y. Tian, J. Ding, H. Chen, Y. Lu, P. Zhou, and W. Zhu, "Five-port optical router for photonic networks-on-chip," *Optics express*, vol. 19, no. 21, pp. 20 258–20 268, 2011.