European Conference on Optical Communication (ECOC) 2022 © Optica Publishing Group 2022

Adapting Routing Algorithms to Programmable Photonic Circuits

Ferre Vanden Kerchove[(1)], Xiangfeng Chen[(2)], Didier Colle[(1)], Wim Bogaerts[(2)], Mario Pickavet[(1)]

[(1)] IDLab, Department of Information Technology, Ghent University - IMEC ferre.vandenkerchove@ugent.be
[(2)] Ghent University - IMEC, Photonic Research Group, Department of Information Technology, Gent, Belgium. Center of Nano and Biophotonics, Ghent University, Belgium

Abstract *The ever-increasing size of programmable photonic integrated circuits necessitates the development of specialised routing algorithms, capable of handling different mesh architectures and magnitudes. We develop an algorithm specifically adapted to the unique characteristics of programmable photonic circuits. ©2022 The Author(s)*

Introduction

Photonic Integrated Circuits (PIC) are the optical equivalent of electronic integrated circuits. Just like their electronical counterparts, PICs are small, energy-efficient and can pack many functions in a single chip. Currently, they are mostly application specific, meaning that they are designed with only one purpose in mind, and the light paths are primarily fixed during chip design. In turn, this leads to long design times and high development costs, possibly slowing down the exploration of novel optical applications[1]. This is where Field-Programmable Photonic Gate Arrays (FPPGAs) come into play. They are made of hundreds of electrically tunable optical couplers, enabling a dynamic control over the course of light. These couplers are systematically arranged in a mesh, as can be seen in Fig. 1. Popular architectures are forward-only meshes, or feedback meshes. This paper focusses on the latter, where light flows in waveguides organized in loops of triangles, squares or hexagonals[2]. Feedback meshes allow light to fold back onto itself, enabling the embedding of a broad class of optical functions, such as interferometers and ring resonators. This reduces the need of dedicated function blocks in FPPGAs.

Right now, the control algorithms to operate these meshes are largely in their infancy. They either take forward-only meshes in mind, or focus on small recirculating meshes. The difficulty of correctly modelling all limitations of light proved to be a real hurdle. However, recent developments[3] have shown an interesting graph representation of couplers, enabling an almost complete separation of the algorithm from the many constraints that are physically imposed. This greatly simplifies algorithm design and permits an easier understanding of the methods involved.

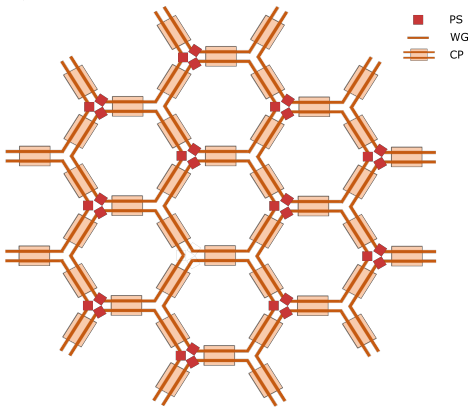

Fig. 1: Waveguides (WG) and phase shifters (PS) organized in hexagonal cells and connected through couplers (CP)

This paper aims to extent the work of[3],[4],[5] focussing on meshes that are an order of magnitude larger. While the aforementioned papers look at meshes consisting of 7 to 20 hexagonals cells, in this paper meshes with hundreds of hexagonal cells are taken into account. We propose several modifications to adapt an algorithm to the characteristics of programmable photonic circuits. The solutions of the resulting algorithm are compared on smaller problem instances where it stays within 2% of the optimal result, with a twentyfold speed up in comparison to the exact solution.

Problem statement

A *commodity* is a source-target pair, where light needs to flow from the source, to the target. The main problem of this paper is the following: given a mesh consisting of waveguides and couplers, and a set of commodities, find a way to connect each source with its target through a path in the mesh. A light wave can only change waveguide by means of a coupler. Such a coupler can let light continue in their waveguide, or cross over to the other waveguide. This is called respectively

Algorithm 1

Require: commodities *coms* and a graph G

$i \leftarrow$ *sol* $\leftarrow 0$
best $\leftarrow \varnothing$
paths $\leftarrow$ PREPROCESSING(G, *coms*)
while $i <$ iter$_{\text{max}}$ **and** *sol* $<$ *sol*$_{\text{max}}$ **do**
  for *com* $\in$ *coms* **do**
    if *paths[com]* **has** *conflict* **then**
      paths[com] $\leftarrow$ DIJKSTRA(G, *com*)
  if NO_CONFLICTS(*paths*) **then**
    best $\leftarrow$ CHOOSE_BEST(*paths*, *best*)
    sol $\leftarrow$ *sol* $+ 1$
    RESET_CONGESTION(G)
    LENGTH_BASED_RIP_UP(G, *paths*)
  else
    UPDATE_WEIGHTS(G, *paths*)
    if DETECT_LOOP(paths, i) **then**
      SEQUENTIAL_ROUTING(G, com)
  $i \leftarrow i + 1$

bar mode and cross mode. Each path needs to adhere to the following constraints: light cannot make a u-turn in a coupler, nor can a coupler be in bar and cross mode at the same time. Additionally, two paths cannot make use of the same waveguides.

Given the earlier mentioned novel graph representation, we can easily model different meshes as graphs. Important to notice is the fact that every waveguide and coupler arm is modelled by two edges, each in one direction. For every edge, we have a *reverse edge*, the edge representing the same waveguide but going in the other direction. The problem now translates to the following. Given a directed graph $G = (V, E)$ and a set of sources-targets pairs $(s_1, t_1), \ldots, (s_n, t_n)$, find a set of paths $P_1, \ldots, P_n$ such that P_i is a path from s_i to t_i and no edge is *congested*. An edge e is considered congested in the following situations.

- Two different paths use e
- A path uses e, another its reverse edge
- A path uses both e and its reverse edge

If a path contains a congested edge, this path is said to have a *conflict*. This definition of congested is broader than normally considered, but a necessary adaptation. A set of paths is a *legal routing* and thus a solution if every source is connected with its target by a conflict-free path.

Algorithm description

We propose a negotiation-based algorithm similar to[6], given the fact that modern FPGA design tools routing algorithms[7] are also built on this idea. We propose several changes, some to incorporate the physical constraints, others to improve performance. Throughout the text, we will call this algorithm Algorithm 1. Different from the original algorithm is the fact that we give weight to the edges instead of to the nodes, to allow for more accurate modelling of intrinsic differences in the waveguides, such as insertion loss (IL), power consumption (P) and basic unit length (BUL). Every edge e has a base weight b_e which is a combination of these three properties, where c_1, c_2, c_3 are the scaling coefficients; $b_e = c_1 \cdot \mathrm{IL}_e + c_2 \cdot \mathrm{BUL}_e + c_3 \cdot P_e$. For simplicity's sake, we will only focus on BUL, and assume that all edges have length exactly 1. Thus, the base weight of every edge is 1.

The *adjusted weight* c_e of an edge e in a path is given by

$$c_e = (b_e + h_e) \cdot p_e.$$

Here b_e is the base weight of an edge as given before, h_e keeps track of how many times that edge was congested. p_e is equal to the amount of other paths that use this edge, plus one. Notice that the weight of an edge can be different, depending on the path. When a least weighted path is calculated by DIJKSTRA[8], it uses these weights. When we refer to *the length of a path*, we refer to the number of edges in that path.

As seen in Algorithm 1, the main loop of the algorithm is fairly similar to Pathfinder[6]. PREPROCESSING will be explained later. The algorithm runs until either a set amount (iter$_{\text{max}}$) of iterations has passed, or a fixed amount (sols$_{\text{max}}$) of legal routings have been found. Every iteration, for every commodity, its corresponding path is ripped-up and rerouted if it has conflicts. The routing happens 'simultaneously', i.e., the routing of a commodity does not depend on what else happened that iteration, only on the weight of that edge, which is only calculated based on the previous iteration.

If no path has any conflicts, the solution is recorded if the sum of the path lengths is shorter than the best solution so far. The factor h_e that keeps track of the congestion is reset on all edges. We want to promote shorter overall path length but keep the good parts of the previously found solution. This is why we only rip-up the paths that are 10% longer than their shortest possible path and then restart again.

If there are still conflicts, every congested edge has its h_e term increased by a fixed amount,

called the *history increase*. We change the history increase every time a solution is found, or when no solution can be found with a particular history increase.

Preprocessing

Right now, the mesh architectures under investigation often have many equal shortest paths between two vertices. PREPROCESSING makes use of this property, by calculating all the shortest paths between a sink and its corresponding target. If two shortest paths between two different source-target pairs both use the same edge, the base cost of that edge is slightly increased by ε, here taken equal to 1% of the base weight. Now, the first round of routing happens. For every commodity, the weighted shortest path is calculated. If a commodity has different shortest paths, it will prefer the shortest path that has no edges shared with a shortest path of another commodity. This reduces the amount of early conflicts but has diminishing effects when the mesh is densely used.

Loop detection

One more problem remains that is not mentioned in context of FPGA routing algorithms. The algorithm gets stuck in a loop, where it keeps suggesting routing R_1, then R_2 and it cycles between these two. This wastes a lot of iteration, and sometimes the algorithm does not even manage to escape from this loop. In one test a cycle of 4 configurations, involving 3 different paths was found. To combat this issue, loop detection is done. If after a set amount of iterations, the same $n \leq 5$ paths still have conflicts and no other paths have, then a round of sequential routing is conducted. This means that after a commodity is routed, weights are immediately updated. Specifically, 10 random sequences are chosen and the shortest conflict-free sequence is chosen, or the sequence with the least amount of conflicts.

Test set generation

A test set consists of a radius and a list of source-target pairs. The mesh is a radial mesh like in Fig. 1. A mesh of radius n has $1 + 3n(n + 1)$ hexagonal cells[1]. The source and targets are randomly placed throughout the outer layers of the mesh.

Results

Fig. 2 plots computational time in comparison to problem size which is the amount of commodities

[1]One single hexagonal is considered a mesh of radius 0.

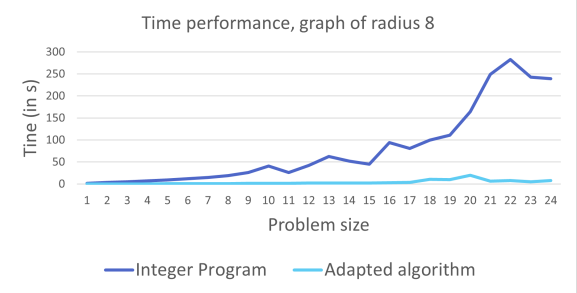

Fig. 2: Time needed for an Integer Program and for Algorithm 1

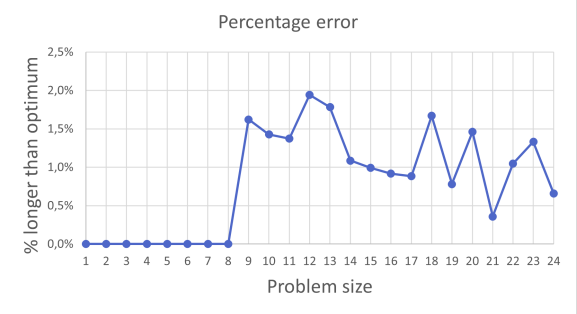

Fig. 3: Total length in comparison to shortest routing

that are routed in a graph of radius 8. As expected, the time needed for an Integer Program to find the optimal solution behaves exponentially in terms of the problem size. For larger graphs, we can no longer find optimal solutions in reasonable time.

To gauge the quality of the solutions of Algorithm 1, we express in percentages how much longer the solution is in comparison to the optimal solution. Fig. 3 shows that on smaller instances the optimal solution is found, while on larger instances the solutions are between 0.5% and 2% longer than the optimum. This is a general trend for all instances still optimally solvable.

Conclusions

We have demonstrated a good routing algorithm specifically adapted to the unique topology of photonic integrated circuits, but independent of exact mesh architecture. Our algorithm produces close-to-optimal results in reasonable time. Future work can consider the embedding of optical functions such as interferometers. This will require more advanced placements techniques that work in tandem with the routing algorithms.

Acknowledgements

Part of this work was funded by the Flemish Research Foundation (FWO-Vlaanderen) through grant G020421 (GRAPHSPAY) and by the European Research Council (ERC) through grant 725555 (PhotonicSWARM).

References

[1] W. Bogaerts and A. Rahim, "Programmable photonics: An opportunity for an accessible large-volume PIC ecosystem", *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 26, no. 5, pp. 1–17, Sep. 2020, Conference Name: IEEE Journal of Selected Topics in Quantum Electronics, ISSN: 1558-4542. DOI: 10.1109/JSTQE.2020.2982980.

[2] D. Pérez, I. Gasulla, J. Capmany, and R. A. Soref, "Reconfigurable lattice mesh designs for programmable photonic processors", *Optics Express*, vol. 24, no. 11, p. 12 093, 2016. DOI: 10.1364/oe.24.012093.

[3] X. Chen, P. Stroobant, M. Pickavet, and W. Bogaerts, "Graph representations for programmable photonic circuits", *Journal of Lightwave Technology*, vol. 38, no. 15, pp. 4009–4018, Aug. 1, 2020, ISSN: 0733-8724, 1558-2213. DOI: 10.1109/JLT.2020.2984990. [Online]. Available: https://ieeexplore.ieee.org/document/9056549/ (visited on 09/24/2021).

[4] X. Chen and W. Bogaerts, "A graph-based design and programming strategy for reconfigurable photonic circuits", in *2019 IEEE Photonics Society Summer Topical Meeting Series (SUM)*, Ft. Lauderdale, FL, USA: IEEE, Jul. 2019, pp. 1–2, ISBN: 978-1-72810-597-0. DOI: 10.1109/PHOSST.2019.8795068. [Online]. Available: https://ieeexplore.ieee.org/document/8795068/ (visited on 09/23/2021).

[5] A. López, D. Pérez, P. DasMahapatra, and J. Capmany, "Auto-routing algorithm for field-programmable photonic gate arrays", *Optics Express*, vol. 28, no. 1, p. 737, 2020. DOI: 10.1364/oe.382753.

[6] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for fpgas", *Third International ACM Symposium on Field-Programmable Gate Arrays*, 1995. DOI: 10.1109/fpga.1995.242049.

[7] K. E. Murray, O. Petelin, S. Zhong, *et al.*, "VTR 8: High-performance CAD and customizable FPGA architecture modelling", *ACM Transactions on Reconfigurable Technology and Systems*, vol. 13, no. 2, pp. 1–55, Jun. 10, 2020, ISSN: 1936-7406, 1936-7414. DOI: 10.1145/3388617. [Online]. Available: https://dl.acm.org/doi/10.1145/3388617 (visited on 10/22/2021).

[8] E. W. Dijkstra, "A note on two problems in connexion with graphs", *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959, ISSN: 0029-599X, 0945-3245. DOI: 10.1007/BF01386390. [Online]. Available: http://link.springer.com/10.1007/BF01386390 (visited on 10/19/2021).