Photontorch: simulation and optimization of large photonic circuits

1st Floris Laporte
Photonics Research Group
Ghent University - imec
Ghent, Belgium
floris.laporte@ugent.be

2nd Joni Dambre
IDLab
Ghent University - imec
Ghent, Belgium
joni.dambre@ugent.be

3rd Peter Bienstman
Photonics Research Group
Ghent University - imec
Ghent, Belgium
peter.bienstman@ugent.be

Abstract—We propose a new framework for simulating and optimizing large photonic circuits. The framework utilizes gpu-acceleration to efficiently simulate the circuits in a highly parallel manner, while optimization is achieved through back-propagation by essentially viewing the network as a sparsely connected recurrent neural network.

Index Terms—photonics, circuits, simulation, optimization, programmable photonics, photonic integrated circuits

I. Introduction

As the field of photonic integrated circuits gradually matures, a trend emerges in which photonic circuits move away from application-specific photonic integrated circuits to more general circuits consisting of well-defined components such as programmable photonic integrated circuits.

Simulating these kind of circuits, however, stays a hard task and definitely has not yet reached the same maturity as for example electronic circuit simulations. This is mostly due to the complex-valued nature of light, where interference effects make sure a large network of identical components acts completely different than the sum of its parts. Additionally, small component variations may propagate through the whole device as interference effects, hampering the design of these circuits even further.

To address these challenges, we present Photontorch [1], a tool that efficiently simulates and optimizes large photonic circuits in time and frequency domain, even with imperfect subcomponents. Photontorch itself is loosely based on the S-matrix approach used by Caphe [2], but is built on top of PyTorch [3], a well-established machine learning library.

The advantages of building a simulation tool on top of such a machine learning library are twofold. First of all, it enables gpu-acceleration. Allowing to efficiently parallelize the simulation. Secondly, PyTorch tracks all operations performed during the simulation, enabling well-established machine learning optimization techniques based on backpropagation.

EU Horizon 2020 PHRESCO Grant (688579); EU Horizon 2020 Fun-COMP Grant (780848); Research Foundation Flanders (FWO) (G024715N);

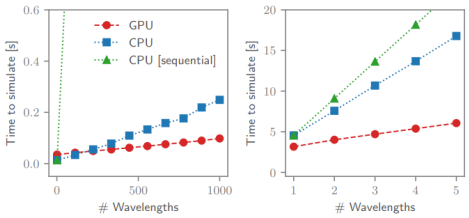

Fig. 1. Simulation times to simulate a CROW circuit in the frequency domain. Left: A CROW with 10 rings was simulated, one can clearly observe a huge benefit to simulating multiple wavelengths simultaneously versus in sequence. Right: simulation times for a CROW with 850 rings. For such large circuits, it is always faster to simulate such as circuit on the GPU, while that difference increases even more if the response to more wavelengths need to be calculated.

II. Performance

To illustrate the power of Photontorch, we choose to quantise its performance by simulating large Coupled Resonator Optical Waveguides (CROW) in the frequency domain. These large circuits are good for benchmarking execution speed, as more rings can easily be added to increase the simulation difficulty. Additionally, A simulation in the frequency domain is an ideal scenario to showcase the performance benefits one can gain by using the parallelized execution of Photontorch. Indeed, as can be seen in Fig. 3, it is clear that simulating many wavelengths simultaneously in stead of in sequence can yield enormous performance benefits, especially when the response to many wavelengths is desired. All simulations were performed on a normal desktop computer with an Intel i7-4790K CPU with 16GB RAM. For the GPU simulations we used an Nvidia GTX-1060 (6GB) GPU.

III. Optimization

The advantages of using Photontorch are however not only performance related. The framework has been written from the ground up in terms of PyTorch tensors, which -

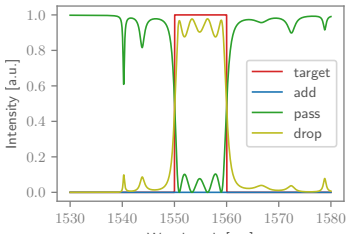

Fig. 2. A CROW can be optimized to act as a bandpass filter in a matter of seconds.

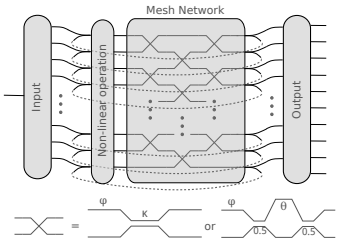

Fig. 3. A three layer mesh circuit with 128 MZIs per layer is foldet onto itself to act add feedback to the system. By adding this additional feedback, a time varying input signal - such as for example an image fed through the input pixel-by-pixel - can be recognized after optimizing the 768 parameters of the mesh network.

as oposed to the more commonly used Numpy ndarrays [4] - track the gradients of each operation performed on them. This creates a dynamic computation graph, which allows the use of backpropagation to optimize the photonic circuits.

As a simple example, we optimize a CROW with 10 rings to act as a bandpass filter between 1550 nm and 1560 nm, as can be seen in Fig. 2. Each ring has a total length of 50 μm. Both the couplings between the rings and the phases in the ring were optimized in a matter of seconds.

Larger circuits can also easily be optimized. As an example, we optimize a mesh-circuit consisting of 384 MZIs (three cascaded layers of each 128 MZIs) to perform the pixel-by-pixel MNIST digit recognition task [5], [6], which is a well-known machine learning benchmark task [7]. Recognizing this time-varying input is possible by connecting the output of the mesh circuit back onto its input, creating a feedback loop. Optimizing the 768 parameters of such a large circuit using conventional circuit simulation tools for a would be a complete nightmare, however it is quite easily done with photontorch. After a few hours of optimizing this network through backpropagation, a final accuracy on the pixel-by-pixel MNIST can be achieved of 92 %, as can be seen in Fig. 4. This is on par with current state-of-the-art recurrent neural network accuracies; achieved by just simulating a photonic circuit.

Note that the choice of application for which to optimize the mesh circuit (in this case the MNIST task) was

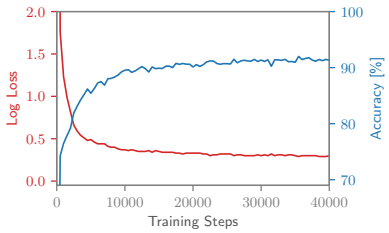

Fig. 4. After a few hours of training the parameters of the photonic circuit, an accuracy on the pixel-by-pixel MNIST task of 92 % can be achieved.

rather arbitrary and just served to show how photontorch performed in optimizing large networks with many parameters.

IV. Conclusion

We proposed a new framework for simulating large photonic circuits, called Photontorch. The framework allows for very fast and parallel simulations by allowing the computations to be placed onto a GPU. This enables very fast multi-wavelength simulations, which can be very useful to calculate the response of a photonic circuit in the frequency domain.

However, the real power of Photontorch lies in its built-in gradient based optimization capabilities that stem from its PyTorch backend. This way, it is possible to view a custom photonic circuit as essentially a sparsely connected recurrent neural network and to use common machine learning optimization techniques such as backpropagation to find the optimal parameters for any circuit.

We showed the usefulness of this approach by providing two examples. One in the frequency domain, where a CROW was optimized in a few seconds to act as a bandpass filter, and one in the time domain, where a large mesh-network was optimized to act as a digit-recognition system.

Acknowledgment

EU Horizon 2020 PHRESCO Grant (688579); EU Horizon 2020 Fun-COMP Grant (780848); Research Foundation Flanders (FWO) (G024715N);

Additional Information

The open-source photontorch framework is available on GitHub: https://github.com/flaport/photontorch.

References

[1] F. Laporte, J. Dambre, and P. Bienstman, “Highly parallel simulation and optimization of photonic circuits in time and frequency domain based on the deep-learning framework pytorch,” Scientific Reports (submitted).

[2] M. Fiers, T. Van Vaerenbergh, K. Caluwaerts, D. V. Ginste, B. Schrauwen, J. Dambre, and P. Bienstman, “Time-domain and frequency-domain modeling of nonlinear optical components at the circuit-level using a node-based approach,” JOSA B, vol. 29, no. 5, pp. 896–900, 2012.

[3] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," Neural Information Processing Systems, 2017.
[4] T. E. Oliphant, A guide to NumPy, vol. 1. Trelgol Publishing USA, 2006.
[5] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," CoRR, 2015.
[6] L. Jing, Y. Shen, T. Dubček, J. Peurifoy, S. Skirlo, Y. LeCun, M. Tegmark, and M. Soljačić, "Tunable efficient unitary neural networks (eunn) and their application to rnns," arXiv preprint arXiv:1612.05231, 2016.
[7] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in Advances in neural information processing systems, pp. 396–404, 1990.