POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

# Graph Neural Network Acceleration with SODA Framework

**Master of Science Thesis in
Computer Science and Engineering**

Author: **Giovanni Demasi**

Student ID: 987062
Advisor: Prof. Fabrizio Ferrandi
Co-advisors: Serena Curzel
Academic Year: 2022-23

# Abstract

Here goes the Abstract in English of your thesis followed by a list of keywords. The Abstract is a concise summary of the content of the thesis (single page of text) and a guide to the most important contributions included in your thesis. The Abstract is the very last thing you write. It should be a self-contained text and should be clear to someone who hasn't (yet) read the whole manuscript. The Abstract should contain the answers to the main scientific questions that have been addressed in your thesis. It needs to summarize the adopted motivations and the adopted methodological approach as well as the findings of your work and their relevance and impact. The Abstract is the part appearing in the record of your thesis inside POLITesi, the Digital Archive of PhD and Master Theses (Laurea Magistrale) of Politecnico di Milano. The Abstract will be followed by a list of four to six keywords. Keywords are a tool to help indexers and search engines to find relevant documents. To be relevant and effective, keywords must be chosen carefully. They should represent the content of your work and be specific to your field or sub-field. Keywords may be a single word or two to four words.

**Keywords:** here, the keywords, of your thesis

# Abstract in Lingua Italiana

Qui va l'Abstract in lingua italiana della tesi seguito dalla lista di parole chiave.

**Parole chiave:** qui, vanno, le parole chiave, della tesi

# Contents

# 1 Introduction

In recent years, deep learning has brought about a revolutionary transformation in various machine learning tasks, spanning from image classification and video processing to speech recognition and natural language understanding. Traditionally, these tasks have predominantly operated within the Euclidean space, where data is typically represented. Nevertheless, a growing number of applications now generate data from non-Euclidean domains, presenting it in the form of complex graphs with intricate relationships and interdependencies among objects. The inherent complexity of graph data has posed considerable challenges for existing machine learning algorithms. Consequently, there has been a surge of studies focusing on extending deep learning techniques to accommodate and leverage graph data.

Graph neural networks (GNNs) have been introduced in response to the growing demand for learning tasks involving graph data, which encompasses extensive relational information among its elements. These neural models effectively capture the interdependence among graph nodes by employing message passing mechanisms.

As Graph Neural Networks are increasingly employed, particularly in domains characterized by vast amounts of data, such as social networks and chemistry, a need arises to optimize and accelerate their capabilities. Inference in GNNs refers to the time the model takes to make predictions after training. The duration of the inference process determines the speed at which queries are answered, and researchers strive to minimize this time span.

In applications of deep learning that prioritize low latency, FPGAs outperform other computing devices, such as CPUs and GPUs, by providing superior performance. FPGAs offer the advantage of being fine-tuned to strike the optimal balance between power efficiency and meeting performance requirements.

Due to this reason, researchers have been actively pursuing the development of new FPGA accelerators for Graph Neural Networks (GNNs) in recent times.

The conventional approach to hardware design involves a combination of manual coding and automated processing. However, this method demands significant effort and relies heavily on the expertise of the designers, leading to varying quality of results.

To address these challenges, the objective of this thesis research study is to develop a comprehensive toolchain that, starting from PyTorch [18], a cutting-edge high-level programming framework for creating neural network algorithms based on the Python programming language, enables the automatic generation of a Graph Neural Networks (GNNs) FPGA accelerator with minimal effort required.

The suggested toolchain represents an enhancement of the SODA toolchain [2]. It operates by transforming the PyTorch model, provided as input, into a multi-level intermediate representation (MLIR) [16] utilizing Torch-MLIR [22], an MLIR based compiler toolkit for PyTorch programs. This MLIR representation is then passed to the SODA framework to conduct hardware/software partitioning of the algorithm specifications and architecture-independent optimizations. Following this, the framework generates a low-level IR (LLVM IR) specifically tailored for the hardware generation engine, PandA-Bambu [9].

In pursuit of the thesis goal, various optimizations were adopted throughout the process. Specifically, efforts were made to optimize specific computations in Graph Neural Networks during the experimental phase. As these networks often deal with massive graph sizes, the computation time and memory requirements are substantial. Consequently, a significant portion of the research focuses on optimizing the computation phase of Graph Neural Networks using tailored SODA optimizations, particularly matrix multiplication.

Furthermore, limitations and challenges have been encountered along the way. Another objective of this thesis is to analyze these limitations, ensuring they are clearly understood thoroughly. This analysis aims to provide valuable insights for future research endeavors, enabling the development of solutions to overcome these limitations and further enhance the proposed toolchain.

While the intended purpose of the toolchain is to be general, the experimental phase primarily focused on two specific types of Graph Neural Networks: Graph Isomorphism Networks (GIN) [24] and Graph Convolutional Networks (GCN) [15]. These models were sourced from reliable GitHub implementations and were modified as necessary.

The GCN model [21], designed for node classification task and written in pure PyTorch, held particular importance for the experimental phase as it served as the basis for the resulting accelerator. On the other hand, the GIN model [20], designed for graph classification task and written in PyTorch Geometric [10], a library built upon PyTorch for easier development and training of Graph Neural Networks, did not progress through the final step of the proposed toolchain. This was due to some incompatibilities between PyTorch Geometric and Torch-MLIR, which are integral parts of this thesis research.

## 1.1 Contributions

## 1.2 Thesis structure

Chapter 1 introduces the context of the thesis, its objective, and its goals, including a general overview of the research's focus, contributions, and outcome. Chapter 2 presents the background needed to understand the thesis's content deeply. In particular, it contains a summary of Graph Neural Networks, how they work, an explanation of the GNN types used in the experimental phase, and the type of tasks that they can perform, including some of their applications. Additionally, it presents the SODA framework, an important part of this thesis's proposed toolchain. Chapter 3 instead contains an overview of the related works. Other Graph Neural Network acceleration frameworks will be analyzed, underlying their differences compared to the research study done for this thesis and some limitations. Chapter 4 formulates the problem statement, summarizes the open issues of the research objective, and explains how the thesis goals can be helpful and their expected impact. Chapter 5 is the core chapter of the thesis, it clearly explains how the problem has been faced and what technologies have been used. It contains a detailed description of the proposed toolchain and its working method. Chapter 6 lists all the performed experiments, gives the necessary information to reproduce them and contains their outcomes and the issues and limitations encountered. Finally, Chapter 7 presents overall considerations of the study, both with the main achievements obtained and the most notable obstacles faced. Along with this, potential improvements for future studies are considered.

# 2 Background

Chapter containing the background needed to understand the problem and its solution, mainly a summary of Graph Neural Networks and SODA toolchain.

## 2.1 Graphs

*Graphs* are a data structure representing a collection of objects, known as vertices or nodes, and a set of edges [26]. In a graph, the edges can be either directed or undirected, and they typically connect two vertices, which may or may not be distinct. The vertices represent entities or elements, and the edges represent their relationships or connections.

Graphs serve as a versatile tool for describing diverse forms of data. Molecules, the fundamental units of matter, are composed of atoms and electrons arranged in three-dimensional space. In this intricate structure, all particles interact with each other. However, when a pair of atoms are stably positioned at a specific distance, we refer to their connection as a covalent bond. These bonds with distinct atomic distances can vary in nature, such as single or double bonds. Representing this complex three-dimensional object as a graph offers a practical and widely adopted abstraction, where atoms are nodes and covalent bonds act as edges [8].

Social networks provide another domain where graphs find utility. They serve as valuable tools for examining patterns within the collective behavior of people, institutions, and organizations. By representing individuals as nodes and their relationships as edges, we can construct a graph that effectively captures groups of people and their interconnectedness.

## 2.2 Graph Neural Networks

Graph neural networks (GNNs) are deep learning techniques that operate on graph-structured data. Thanks to their impressive performance, GNNs have recently gained significant popularity as a widely adopted method for graph analysis.

Graph Neural Networks (GNNs) are designed to process graph data and consist of multiple interconnected layers. Each GNN layer typically encompasses three main stages: feature extraction, aggregation, and updating, with an optional sampling stage. The feature extraction is employed to compress vertex features using processing functions like MLPs (Multi-Layer Perceptrons) and, together with the update stage, resembles traditional neural network inference, involving regular computational operations and memory access patterns. The aggregation stage traverses the graph topology to combine features from a vertex's neighboring vertices and can be accomplished using functions like sum, mean, or max. The update stage is utilized to apply non-linear transformations, including activation functions, GRU (Gated Recurrent Units), and MLPs, to the graph data. Finally, the optional sampling stage constructs a new graph through user-defined sampling operations. Performing these stages on large and sparse graphs can introduce dynamic computational data flow and numerous irregular memory access patterns.

Graph Neural Networks are a group of neural networks which are designed to solve different tasks. Prediction tasks on graphs can generally be classified into three categories: graph-level, node-level, and edge-level predictions [19].

In a graph-level task, the objective is to predict the property or characteristic of an entire graph. For instance, when considering a molecule represented as a graph, we might aim to predict attributes such as its likelihood of binding to a receptor associated with a specific disease. This assignment is comparable to image classification tasks, where the objective is to assign a label to an entire image. Similarly, in text analysis, sentiment analysis serves as a similar problem where the goal is to determine a complete sentence's overall mood or emotion in one go.

Node-level tasks involve predicting the identity or function of individual nodes within a graph. One example of a node-level task is node classification in a social network. Given a social network graph where nodes represent individuals and edges represent relationships between them, the task is to predict the demographic attributes or characteristics (e.g., age, gender, occupation) of each node based on their connection patterns and features. Drawing an analogy to image processing, node-level prediction problems can be compared to image segmentation tasks, where the objective is to assign labels to each pixel in an image based on its role. Similarly, in text analysis, a comparable task would involve predicting the parts of speech for each word in a sentence, such as identifying whether a word is a noun, verb, adverb, and so on.

The remaining prediction task in graphs pertains to edge prediction. One example of an edge-level task is link prediction in a social network. Given a graph representing a social network where, as before, in node-level tasks, nodes correspond to individuals and edges represent relationships between them, the edge-level task aims to predict missing or potential connections between nodes. This can involve predicting the likelihood of a future friendship or the probability of a collaboration between individuals based on their shared characteristics or mutual connections in the network.

Different popular Graph Neural Network architectures have been proposed recently, some of which are more suitable for some tasks than others. A summary of two types of GNNs used in the experimental phase is provided in the following sections.

### 2.2.1 Graph Convolutional Network

A graph convolutional network (GCN) [7, 15] is a type of neural network architecture explicitly designed to operate on graph-structured data. GCNs aim to learn node representations by aggregating and combining information from neighboring nodes in the graph. The core idea behind GCNs is to perform convolution-like operations on the graph, where the convolutional filters are defined based on the graph's adjacency matrix or other graph-specific structures. This enables GCNs to capture and leverage the structural information encoded in the graph to make predictions or perform downstream tasks. GCNs have demonstrated effectiveness in various applications, including node classification, link prediction, and graph classification.

Given an undirected graph $\mathcal{G} = (V, E)$, where $V$ represents the set of nodes (vertices), and $E$ represents the set of edges, with an adjacency matrix $\tilde{A} = A + I_N$, where $I_N$ is the identity matrix, the layer-wise propagation rule in a GCN can be expressed as:

$$H^{(l+1)} = f(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{2.1}$$

Where $H^{(l)} \in \mathbb{R}^{N \times D}$ is the input node features matrix, $W^{(l)}$ is a layer-specific learnable weight matrix, $\tilde{D}$ is the degree matrix defined as $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and $f(\cdot)$ represents a non-linear activation function applied element-wise, such as $ReLU(\cdot) = max(0, \cdot)$. The equation above demonstrates the propagation of node features through graph convolution, where the adjacency matrix $\tilde{A}$ captures the connectivity information of the graph, $\tilde{D}^{-\frac{1}{2}}$ normalizes the adjacency matrix, and $H^{(l)}W^{(l)}$ performs a linear transformation of node features. The resulting $H^{(l+1)}$ represents the updated node representations after the graph convolution operation. In practice, multiple graph convolutional layers can be stacked to capture increasingly complex relationships and refine the node representations further.

### 2.2.2 Graph Isomorphism Network

A Graph Isomorphism Network (GIN) [7, 24] is a type of neural network architecture designed to operate on graph-structured data by capturing graph isomorphism, which is the property of two graphs having the same structure, inspired by the Weisfeiler-Lehman (WL) graph isomorphism test [24]. GINs aim to learn node representations that are invariant under graph isomorphism, enabling them to generalize across different graphs with similar structures.

The learned vertex features from GIN-Conv can be directly utilized for tasks such as node classification and link prediction. It is possible to perform this model as:

$$h_v^{(k+1)} = MLP^{(k)}\left(\left(1+\epsilon^{(k)}\right)\cdot h_v^{(k)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k)}\right) \tag{2.2}$$

Where $h_v^{(k)}$ represents the initial node representation of node $v$, $\mathcal{N}(v)$ represents the neighborhood of node $v$, $\epsilon$ is a learnable parameter or a fixed scalar, $MLP(\cdot)$ represents a Multi Layer Perceptron and $h_v^{(k+1)}$ represents the updated node representations.

In the neighborhood aggregation process of GINs, each node's representation is updated by considering its own representation and its neighbors' representations. The neighborhood aggregation is performed through the MLP operation, followed by non-linear activation.

GINs are trained using graph-level objectives, such as graph classification or property prediction, and aim to learn invariant representations under graph isomorphism, allowing them to generalize well to unseen graphs with similar structures. However, even if the node embeddings acquired through GIN can be directly applied to tasks such as node classification and link prediction, in the case of graph classification tasks, it is necessary to use a Readout function that takes individual node embeddings as input and produces the embedding representation for the entire graph.

The Readout function is then utilized to generate the overall representation of the graph, leveraging the individual vertex representations. By concatenating the results from all iterations of GINConv, the final graph representation is obtained as:

$$h_G = CONCAT\left(READOUT\left(\left\{h_v^{(k)} | v \in G\right\}\right) | k = 0, 1, ..., K\right) \tag{2.3}$$

Where $READOUT$ in 2.2 can be replaced with a sum operator in order to generalize the WL test [24].

## 2.3 SODA Toolchain

# 3 Related Work

Analysis of related works, explaining how other GNN accelerators have been built, and some limitations of their approach.

# 4 Problem Formulation

Problem formulated in a clear way, what we did and how, with open issues and thesis goals.

# 5 FPGA Toolchain for Graph Neural Network Acceleration

Introduction of the way I faced the problem, with the motivation for the followed approach. Explanation of the toolchain in a clear way.

# 6 Experimental Results

Chapter dedicated to the outcome of the results, what I have obtained and what limitations have been encountered. Explaining the still open issues and research suggestions.

# 7 Conclusions and Future Developments

Final chapter containing the main conclusions of my research and possible future developments.

# Bibliography

[1] S. Abi-Karam, Y. He, R. Sarkar, L. Sathidevi, Z. Qiao, and C. Hao. Gengnn: A generic FPGA framework for graph neural network acceleration. *CoRR*, abs/2201.08475, 2022. URL `https://arxiv.org/abs/2201.08475`.

[2] N. B. Agostini, S. Curzel, J. J. Zhang, A. Limaye, C. Tan, V. Amatya, M. Minutoli, V. G. Castellana, J. Manzano, D. Brooks, G.-Y. Wei, and A. Tumeo. Bridging python to silicon: The soda toolchain. *IEEE Micro*, 42(5):78–88, 2022. doi: 10.1109/MM.2022.3178580.

[3] A. Auten, M. Tomei, and R. Kumar. Hardware acceleration of graph neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020. doi: 10.1109/DAC18072.2020.9218751.

[4] A. Bik, P. Koanantakool, T. Shpeisman, N. Vasilache, B. Zheng, and F. Kjolstad. Compiler support for sparse tensor computations in MLIR. *ACM Transactions on Architecture and Code Optimization*, 19(4):1–25, sep 2022. doi: 10.1145/3544559. URL `https://doi.org/10.1145%2F3544559`.

[5] U. Bondhugula. High performance code generation in MLIR: an early case study with GEMM. *CoRR*, abs/2003.00532, 2020. URL `https://arxiv.org/abs/2003.00532`.

[6] S. Böhm. How to optimize a cuda matmul kernel for cublas-like performance: a worklog, 2022. URL `https://siboehm.com/articles/22/CUDA-MMM`.

[7] A. Daigavane, B. Ravindran, and G. Aggarwal. Understanding convolutions on graphs. *Distill*, 2021. doi: 10.23915/distill.00032. https://distill.pub/2021/understanding-gnns.

[8] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *CoRR*, abs/1509.09292, 2015. URL `http://arxiv.org/abs/1509.09292`.

[9] F. Ferrandi, V. G. Castellana, S. Curzel, P. Fezzardi, M. Fiorito, M. Lattuada,

M. Minutoli, C. Pilato, and A. Tumeo. Invited: Bambu: an open-source research framework for the high-level synthesis of complex applications. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1327–1330, 2021. doi: 10.1109/DAC18074.2021.9586110.

[10] M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *CoRR*, abs/1903.02428, 2019. URL `http://arxiv.org/abs/1903.02428`.

[11] L. He. Engn: A high-throughput and energy-efficient accelerator for large graph neural networks. *CoRR*, abs/1909.00155, 2019. URL `http://arxiv.org/abs/1909.00155`.

[12] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22118–22133. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/fb60d411a5c5b72b2e7d3527cfc84fd0-Paper.pdf`.

[13] Y. Hu, Y. Du, E. Ustun, and Z. Zhang. Graphlily: Accelerating graph linear algebra on hbm-equipped fpgas. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021. doi: 10.1109/ICCAD51958.2021.9643582.

[14] K. Kiningham, C. Ré, and P. A. Levis. GRIP: A graph neural network accelerator architecture. *CoRR*, abs/2007.13828, 2020. URL `https://arxiv.org/abs/2007.13828`.

[15] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL `http://arxiv.org/abs/1609.02907`.

[16] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko. Mlir: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14, 2021. doi: 10.1109/CGO51591.2021.9370308.

[17] S. Liang, C. Liu, Y. Wang, H. Li, and X. Li. Deepburning-gl: an automated framework for generating graph neural network accelerators. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2020.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison,

A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. URL `http://arxiv.org/abs/1912.01703`.

[19] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. doi: 10.23915/distill.00033. https://distill.pub/2021/gnn-intro.

[20] O. G. B. team and other contributors. Gnn models from open graph benchmark, 2020. URL `https://github.com/snap-stanford/ogb/tree/master/examples/graphproppred/mol`.

[21] M. W. Thomas N. Kipf and other contributors. Gcn model in pytorch, 2017. URL `https://github.com/tkipf/pygcn`.

[22] n. t. Torch-MLIR team and other contributors. Torch-mlir: Mlir based compiler toolkit for pytorch programs, 2021. URL `https://github.com/llvm/torch-mlir`.

[23] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019. URL `http://arxiv.org/abs/1901.00596`.

[24] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks?, 2019.

[25] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie. Hygcn: A GCN accelerator with hybrid architecture. *CoRR*, abs/2001.02514, 2020. URL `http://arxiv.org/abs/2001.02514`.

[26] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018. URL `http://arxiv.org/abs/1812.08434`.

# List of Figures

# List of Tables

# List of Symbols

| **Variable** | **Description** | **SI unit** |
|---|---|---|
| $\boldsymbol{u}$ | solid displacement | m |
| $\boldsymbol{u}_f$ | fluid displacement | m |

# Acknowledgements

Acknowledgements here...