# High-Performance Computing for Graph AI: A Top-Down Perspective

Yuede Ji
*Department of Computer Science and Engineering*
*University of Texas at Arlington*

***Abstract*—A graph, made up of vertices and edges, is a natural representation for many real-world applications. Graph artificial intelligence (AI) techniques, especially graph neural networks (GNNs), are becoming increasingly important in modern machine learning and data analysis, as they can accurately represent high-dimensional features of vertices, edges, and structure information into low-dimensional embeddings. They have become a valuable area of study for students in fields like computer science, data science, and AI. However, the students are facing two challenges to grasp the knowledge of GNNs, including (i) *learning GNNs often requires multidiscipline knowledge*, and (ii) *resources for learning GNNs are often fragmented*.**

**Motivated by that, we designed a self-contained course module on *high-performance computing for graph AI: from a top-down perspective* based on our study in this area for the past years. In particular, we divide them into four levels from the top to the bottom, including (i) level 1: graph theory basics, (ii) level 2: fundamental theories of GNNs, (iii) level 3: efficient graph AI computation framework, and (iv) level 4: GPU architecture and programming. In addition, we have disseminated part of this module into different educational activities, such as courses and tutorials.**

***This paper is submitted for the Research to Education track of EduPar-25.***

## I. Introduction

A graph, made up of vertices (or nodes) and edges, is a natural representation for many real-world applications, such as social network [1], [2], [3], road map [4], and computer network [5]. In recent years, there has been a great surge of research interest on graph artificial intelligence (AI), especially graph neural networks (GNNs), which mainly use message-passing mechanisms to aggregate information from a vertex's neighbors and update its representation iteratively.

GNNs can accurately represent high-dimensional features of vertices, edges, and structure information into low-dimensional embeddings [6], that can be further utilized for various down-stream tasks, such as vertex classification [7], [8], graph classification [9], and link prediction [10], [11]. Because of that, GNNs have been applied in various real-world applications, such as estimated time of arrival (ETA) predication in Google Maps [12], protein structure prediction in Alphafold [13], friend recommendation in LinkedIn [14], and code vulnerability detection in cybersecurity [15].

As GNNs are becoming increasingly important in modern machine learning and data analysis, they have become a valuable area of study for students in fields like computer science, data science, and AI. However, the students are facing two challenges to grasp the knowledge of GNNs. *(i) Learning GNNs often requires multidiscipline knowledge*, including graph theory, AI, parallel and distributed computation, and GPU. In addition, the students need to learn domain-specific knowledge if they want to deploy GNNs to specific applications, e.g., protein analysis. *(ii) Resources for learning GNNs are often fragmented* across research papers, online tutorials, and technical blogs, making it hard to follow a cohesive learning path.

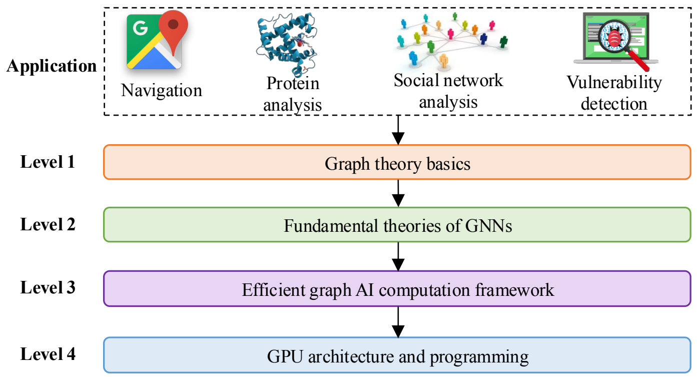

Fig. 1: Overview of the proposed module of *HPC for graph AI from a top-down perspective*.

Motivated by that, we designed a self-contained course module on *high-performance computing for graph AI: from a top-down perspective* based on our study in this area for the past ten years, including graph theory [16], [17], [18], [19], high-performance computation (HPC) for GNN [20], [21], and deploying graph and GNNs in real applications [22], [23], [24], [25], [15], [26], [27], [28].

Figure 1 presents an overview of the proposed module. In particular, we divide them into four levels from the top to the bottom, including (i) level 1: graph theory basics, (ii) level 2: fundamental theories of GNNs, (iii) level 3: efficient graph AI computation framework, and (iv) level 4: GPU architecture and programming. We refer the top to the graph theory basics because we believe they are the essentials of graph AI and should be the entry-level knowledge for learning GNNs. Then, we discuss the fundamental theories of GNNs. Next, we introduce two efficient GNN computation frameworks. Lastly, we refer the bottom to the low-level computation on the GPU architecture, which could help the students understand what actually happened during runtime.

**Educational activities.** We have integrated part of this module into several educational activities. In particular, we created a new course Advanced Topics in CSE concentrating on graph theory and GNN. I have taught this course for Spring 2023 and Spring 2024. In addition, I have taught Analysis of Computer Algorithms for three semesters, where I deeply discussed graph theory-related topics. I also disseminate part of this module into a tutorial on "Graph Algorithms" at *Digital Divas 2023 to female high school students* in the state of Texas [29].

In summary, this study makes two major contributions.

- **A new perspective of teaching graph AI**. We designed a self-contained course module on *high-performance computing for graph AI: from a top-down perspective*.
- **Disseminating into courses and tutorials.** We disseminated part of this module into several educational activities, including both courses and tutorials.

## II. Methodology

This section briefly discusses the contents of the designed module on HPC for GNN from the top-down perspective.

### *A. Level 1: Graph Theory Basics*

At the first level, we discuss the basics of graph theory, which are the fundamentals for understanding the more advanced concepts in GNNs. We will cover two major topics, including *(i)* graph basics, and *(ii)* fundamental graph algorithms.

**Graph basics.** A graph can be represented as $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges, and $|V|$ and $|E|$ denote the number of vertices and edges in the graph, respectively. Following that, we discuss different types of graphs, including undirected/directed graphs, and unweighted/weighted graphs. Next, we discuss the basic graph properties, including degrees, (simple) paths, and subgraphs.

**Fundamental graph algorithms.** In this part, we discuss two major types of graph algorithms, i.e., graph traversal, and graph connectivity. *(i) Graph traversal* is a fundamental method in graph theory. It visits all the vertices and edges of a graph systematically, whose goal is to explore a graph. Because of that, it often serves as a fundamental method for other graph algorithms like searching for a specific vertex, finding the shortest path, or analyzing connectivity. We cover both depth-first search (DFS) and breadth-first search (BFS). In short, DFS explores a graph by starting at a source vertex and diving as deep as possible into one branch before backtracking. It uses a stack to track vertices, prioritizing unvisited neighbors of the current vertex. Differently, BFS explores a graph layer by layer, visiting all neighbors of a vertex before moving forward. It uses a queue to track vertices, ensuring vertices closer to the source are visited first.

*(ii) Graph connectivity* algorithms answer whether two nodes in a graph are connected under certain conditions. In an undirected graph, the graph is considered connected if there is a path between every pair of vertices. In a directed graph, the graph can be strongly connected (if there is a directed path between every pair of vertices) or weakly connected (if replacing all directed edges with undirected ones results in a connected graph). Connectivity is crucial in applications like network design and reliability analysis.

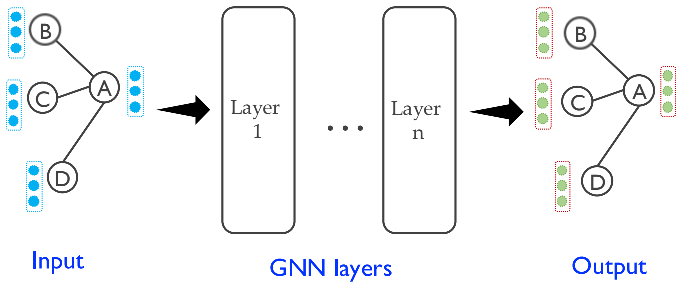

Fig. 2: Basic architecture of GNNs.

### *B. Level 2: Fundamental Theories of GNNs*

After understanding the basics of graph and graph theory, we move to the discussion of GNNs. At this level, we discuss the GNN basics and their unique computation patterns.

**GNN basics.** Figure 2 shows the basic architecture of GNNs. A GNN usually takes an attributed graph as input, which is the combination of the graph structure and vertex features. The goal of GNN is to learn a representation for each vertex, which can further be optimized for graph or edge representations. Typically, existing GNNs follow a neighborhood aggregation strategy, where it iteratively updates a vertex's representation by aggregating the representations of its neighbors and itself. We denote a vertex $v$'s representation in the $k$-th layer as $\boldsymbol{h}_v^{(k)}$, with $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$, and its neighbor set as $\mathcal{N}(v)$. Then, GNN learns $v$'s representation at the $k$-th layer as:

$$\boldsymbol{l}_v^{(k)} = \text{Aggregate}\big(\{\boldsymbol{h}_u^{(k-1)} : u \in \mathcal{N}(v)\}\big) \tag{1}$$

$$\boldsymbol{h}_v^{(k)} = \text{Update}\big(\boldsymbol{h}_u^{(k-1)}, \boldsymbol{l}_v^{(k)}\big) \tag{2}$$

Different GNNs define different aggregate and update functions. With that, we discuss two representative GNNs, i.e., Graph Convolutional Network (GCN) [30] and Graph Attention Network (GAT) [31]. For example, in GCN [30], the aggregate and update functions are defined as:

$$\boldsymbol{h}_v^{(k)} = \text{ReLU}\big(\boldsymbol{W}^{(k)} \cdot \text{MEAN}\{\boldsymbol{h}_u^{(k-1)} : u \in \mathcal{N}(v) \bigcup \boldsymbol{h}_v^{(k-1)}\}\big) \tag{3}$$

$$= \text{ReLU}\Big(\sum_{u \in \mathcal{N}(v)} \boldsymbol{W}^{(k)} \boldsymbol{h}_u^{(k-1)} + \boldsymbol{W}^{(k)} \boldsymbol{h}_v^{(k-1)}\Big) \tag{4}$$

**GNN computation patterns.** After understanding the theories of GNNs, we dive deeper from the computation perspective. In particular, we look deep into the computation of one GNN layer, which mainly follows a three-phase pattern. *(i)* The features are processed by some regular neural operations, e.g., *Dropout*, and *Matmul*. *(ii)* Next, all the features are fed into a phase called *Graph Convolution*, in which each vertex aggregates the features of all its neighbors and the associated edges, applies neural operations, combines the results with its own features, and uses a reduce operation (e.g. *max*, *mean*) to produce a new feature vector. *(iii)* Then, the features are usually applied by operations such as activation function, batch normalization, and softmax, similar to traditional DNNs. Then, the newly generated features will be passed to the next layer.

### C. Level 3: Efficient Graph AI Computation Framework

After understanding the fundamentals of GNNs, we would like the students to get hands-on experiences of running GNNs with existing efficient GNN computation frameworks. At this level, we introduce two well-maintained frameworks, i.e., PyTorch Geometric (PyG) [32], and Deep Graph Library (DGL) [33].

PyG is a Python library built on PyTorch that provides tools and modules for implementing GNNs [32]. It is designed to facilitate deep learning on graph-structured data by offering efficient operations, predefined models, and utilities for handling graphs.

Similarly, DGL is a Python-based, open-source framework for building and training GNNs [33]. It is designed to be scalable, flexible, and efficient, enabling researchers and developers to process and analyze graph-structured data using neural network techniques.

With either PyG or DGL, a GNN could be simply implemented in just a few lines of code. In addition, they have integrated multiple publicly available graph datasets, which can be easily tested.

### D. Level 4: GPU Architecture and Programming

At the bottom level, we would like to discuss the GPU architecture and programming, which served as the backend for GNN computation frameworks. In this part, we use Nvidia GPUs as representatives. We will discuss GPU architecture, GPU programming, and GPU profiling.

**GPU architecture.** Figure 3 shows a brief overview of the mapping between the CUDA (Compute Unified Device Architecture) programming model and the underlying GPU hardware. A grid maps to the whole GPU device. Then, inside a grid, there are many blocks. From the hardware view, the computation of one or multiple blocks are handled by one streaming multiprocessor (SM). Inside each block, it can run many threads. The threads are running in the CUDA cores of an SM. Also, there is warp inside a block. It usually has 32 threads. There are also registers, and all the threads in one block share the same L1 cache and configurable shared memory. Further, the whole device shares the same L2 cache and global memory.

**GPU programming.** CUDA-based GPU programming allows users to leverage the parallel processing power of NVIDIA GPUs for high-performance computing tasks [34]. CUDA is a parallel computing platform and API that provides direct access to the GPU's virtual instruction set and memory. It enables programmers to write code in C, C++, or Python (via libraries like PyCUDA and Numba) to execute highly parallel

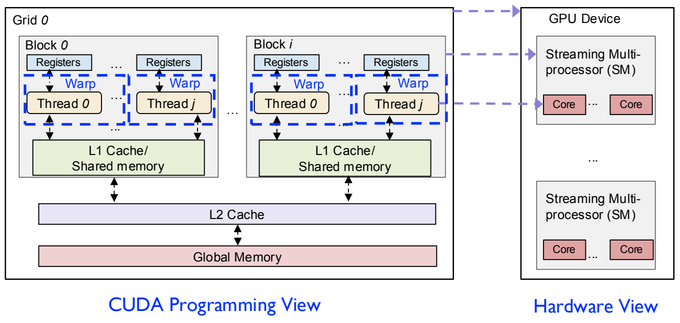

Fig. 3: An overview of the GPU architecture and CUDA programming model.

tasks efficiently. CUDA programming involves dividing tasks into smaller threads, organized into blocks and grids, which run simultaneously on the GPU. In this part, we provide a hands-on tutorial on CUDA programming with use cases of GNNs.

**GPU profiling.** GPU profiling is the process of analyzing and optimizing the performance of applications running on a GPU. Profiling tools, such as NVIDIA's Nsight Systems and Nsight Compute [35], provide insights into various metrics like kernel execution time, memory utilization, and data transfer between the CPU and GPU. These tools help understand the behaviors and identify bottlenecks in parallel execution, inefficient memory access patterns, and underutilized hardware resources. By examining these metrics, users can have a better idea of what is under the hood. Also, they can fine-tune their code to improve performance, such as optimizing thread usage, minimizing warp divergence, and ensuring efficient memory coalescing. GPU profiling is an essential step in developing high-performance GPU applications, enabling users to maximize computational throughput and achieve better scalability for intensive workloads. In this part, we provide a hands-on tutorial on profiling GPUs.

## III. Discussion and Conclusion

In this work, we designed a self-contained course module on *high-performance computing for graph AI: from a top-down perspective*. In particular, we divide them into four levels from the top to the bottom, including (i) level 1: graph theory basics, (ii) level 2: fundamental theories of GNNs, (iii) level 3: efficient graph AI computation framework, and (iv) level 4: GPU architecture and programming. In addition, we have disseminated part of this module into different educational activities, such as courses and tutorials.

## Acknowledgment

This work was supported in part by National Science Foundation grants 2331301, 2409211, 2319975, 2419843. The views, opinions, and/or findings expressed in this material are those of the authors and should not be interpreted as representing the official views of the National Science Foundation, or the U.S. Government.

## References

[1] J. Scott, "Social network analysis," *Sociology*, 1988.

[2] Y. Ji, Y. He, X. Jiang, J. Cao, and Q. Li, "Combating the evasion mechanisms of social bots," *Computers & Security*, 2016.

[3] J. Cao, Q. Li, Ji, Yuede, Y. He, and D. Guo, "Detection of forwarding-based malicious urls in online social networks," *International Journal of Parallel Programming*, vol. 44, no. 1, pp. 163–180, 2016.

[4] L. Cao and J. Krumm, "From gps traces to a routable road map," *Proceedings of ACM SIGSPATIAL*, 2009.

[5] K. L. Calvert, M. B. Doar, and E. W. Zegura, "Modeling internet topology," *IEEE Communications magazine*, 1997.

[6] Z. Jia, S. Lin, M. Gao, M. Zaharia, and A. Aiken, "Improving the accuracy, scalability, and performance of graph neural networks with roc," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 187–198, 2020.

[7] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[8] L.-k. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process." in *AAAI*, 2018.

[9] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[10] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in Neural Information Processing Systems*, vol. 31, pp. 5165–5175, 2018.

[11] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, and C. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 5363–5370.

[12] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire *et al.*, "Eta prediction with graph neural networks in google maps," in *Proceedings of the 30th ACM international conference on information & knowledge management*, 2021, pp. 3767–3776.

[13] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[14] F. Borisyuk, S. He, Y. Ouyang, M. Ramezani, P. Du, X. Hou, C. Jiang, N. Pasumarthy, P. Bannur, B. Tiwana *et al.*, "Lignn: Graph neural networks at linkedin," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 4793–4803.

[15] Y. Ji, L. Cui, and H. H. Huang, "BugGraph: Differentiating Source-Binary Code Similarity with Graph Triplet-Loss Network," in *16th ACM ASIA Conference on Computer and Communications Security (AsiaCCS)*, 2021.

[16] W. Feng, S. Chen, H. Liu, and Y. Ji, "Peek: A prune-centric approach for k shortest path computation," in *SC*, 2023.

[17] Y. Ji and H. H. Huang, "Aquila: Adaptive parallel computation of graph connectivity queries," in *Proceedings of HPDC*, 2020.

[18] Y. Ji, H. Liu, and H. H. Huang, "ispan: Parallel identification of strongly connected components with spanning trees," in *Proceedings of SC*. IEEE, 2018.

[19] ——, "SwarmGraph: Analyzing Large-Scale In-Memory Graphs on GPUs," in *International Conference on High Performance Computing and Communications (HPCC)*. IEEE, 2020.

[20] Q. Fu, Y. Ji, and H. H. Huang, "Tlpgnn: A lightweight two-level parallelism paradigm for graph neural network computation on gpu," in *HPDC*, 2022.

[21] S. Chen, D. Zheng, C. Ding, C. Huan, Y. Ji, and H. Liu, "Tango: Re-thinking quantization for graph neural network training on gpus," in *SC*, 2023.

[22] H. He, X. Lin, Z. Weng, R. Zhao, S. Gan, L. Chen, Y. Ji, J. Wang, and Z. Xue, "Code is not natural language: Unlock the power of semantics-oriented graph representation for binary code similarity detection," in *The 33rd USENIX Security Symposium (USENIX Security)*, 2024.

[23] L. Cui, J. Cui, Y. Ji, Z. Hao, L. Li, and Z. Ding, "Api2vec: Learning representations of api sequences for malware detection," in *International Symposium on Software Testing and Analysis (ISSTA)*, 2023.

[24] H. He, Y. Ji, and H. H. Huang, "Illuminati: Towards Explaining Graph Neural Networks for Cybersecurity Analysis," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2022.

[25] Y. Ji and H. H. Huang, "NestedGNN: Detecting Malicious Network Activity with Nested Graph Neural Networks," in *IEEE International Conference on Communications (ICC)*, 2022.

[26] B. Bowman, C. Laprade, Y. Ji, and H. H. Huang, "Detecting lateral movement in enterprise computer networks with unsupervised graph ai," in *Proceedings of RAID*, 2020.

[27] Y. Ji, L. Cui, and H. H. Huang, "Vestige: Identifying Binary Code Provenance for Vulnerability Detection," in *International Conference on Applied Cryptography and Network Security (ACNS)*. Springer, 2021, pp. 287–310.

[28] Y. Ji, M. Elsabagh, R. Johnson, and A. Stavrou, "DEFInit: An Analysis of Exposed Android Init Routines," in *30th USENIX Security Symposium (USENIX Security)*, 2021.

[29] "Digital divas 2023," https://digital-divas.weebly.com/.

[30] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ICLR*, 2017.

[31] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[32] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[33] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai *et al.*, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.

[34] M. Harris, "Cuda 9 features revealed: Volta, cooperative groups and more," 2017. [Online]. Available: https://devblogs.nvidia.com/cuda-9-features-revealed/

[35] Nvidia, "Nvidia nsight compute," Oct 2021. [Online]. Available: https://developer.nvidia.com/nsight-compute