Possible Reference 1:

Malagon, Edwin, and Alexis Rojas. “Analysis and Simulation of Graphs Applied to Learning with Parallel Programming in HPC.” *2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON), Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON), 2017 CHILEAN Conference On*, Oct. 2017, pp. 1–7. *EBSCOhost*, doi:10.1109/CHILECON.2017.822964

“Large-scale graph analysis or also called network analysis of networks is supported by different algorithms, among the most relevant are PageRank (Web page ranking), Betweenness centrality (centrality in a graph) and Community Detection, these by of their complexity and the large amount of data that process diverse applications, increasingly need to use computational resources such as processor, memory and storage, for these reasons, it is necessary to apply high performance computing or HPC (High Performance Computing) but it would not be useful to apply HPC without having designed these algorithms in parallel programming, in this part there have been many studies on its application and methodologies to do it. The purpose of this work is to create a framework that allows computer science students to abstract a computer system based on the parallel programming paradigm, which implies that students to get acquainted with the resolution of algorithmic problems in a more natural way and away from the typical sequential thinking., The development of a graph analysis design pattern oriented to parallel programming in HPC, complemented with the design of didactic learning techniques in the network such as laboratories and/or simulators are key in the development of this framework.”

Possible Reference 2:

Kogge, Peter M., et al. “Introducing Streaming into Linear Algebra-Based Sparse Graph Algorithms.” *2019 International Conference on High Performance Computing & Simulation (HPCS), High Performance Computing & Simulation (HPCS), 2019 International Conference On*, July 2019, pp. 486–495. *EBSCOhost*, doi:10.1109/HPCS48598.2019.9188143.

“GraphBLAS is a new package designed to provide a standard set of building blocks for graph algorithms based formally in the language of linear algebra. This paper suggests some extensions of the underlying math that would enhance GraphBLAS’ ability to stream updates into a computation without a bulk recomputation, and at greatly reduced computational complexity. The process is applied to several examples.”

Possible Reference 3:
Barrett, Richard F., et al. “Exploring Chapel Productivity Using Some Graph Algorithms.” *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2020 IEEE International*, May 2020, p. 672. *EBSCOhost*, doi:10.1109/IPDPSW50202.2020.00114.

“A broad set of data science and engineering questions may be organized as graphs, providing a powerful means for describing relational data. Although experts now routinely compute graph algorithms on huge, unstructured graphs using high performance computing (HPC) or cloud resources, this practice hasn’t yet broken into the mainstream. Such computations require great expertise, yet users often need rapid prototyping and development to quickly customize existing code. Toward that end, we are exploring the use of the Chapel programming language as a means of making some important graph analytics more accessible, examining the breadth of characteristics that would make for a productive programming environment, one that is expressive, performant, portable, and robust.”

Possible Reference 4:
Slota, G. M. (. 1.)., et al. “Scalable Generation of Graphs for Benchmarking HPC Community-Detection Algorithms.” *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*. *EBSCOhost*, doi:10.1145/3295500.3356206. Accessed 11 Oct. 2020.

“Community detection in graphs is a canonical social network analysis method. We consider the problem of generating suites of teras-cale synthetic social networks to compare the solution quality of parallel community-detection methods. The standard method, based on the graph generator of Lancichinetti, Fortunato, and Radicchi (LFR), has been used extensively for modest-scale graphs, but has inherent scalability limitations. We provide an alternative, based on the scalable Block Two-Level Erdos-Renyi (BTER) graph generator, that enables HPC-scale evaluation of solution quality in the style of LFR. Our approach varies community coherence, and retains other important properties. Our methods can scale real-world networks, e.g., to create a version of the Friendster network that is 512 times larger. With BTER's inherent scalability, we can generate a 15-terabyte graph (4.6B vertices, 925B edges) in just over one minute. We demonstrate our capability by showing that label-propagation community-detection algorithm can be strong-scaled with negligible solution-quality loss. © 2019 ACM.”

Possible Reference 5:
Goodarzi, Bahareh, et al. “High Performance Multilevel Graph Partitioning on GPU.” *2019 International Conference on High Performance Computing & Simulation (HPCS), High Performance Computing & Simulation (HPCS), 2019 International Conference On*, July 2019, pp. 769–778. *EBSCOhost*, doi:10.1109/HPCS48598.2019.9188120.

“Graph partitioning is a common computational phase in many application domains, including social network analysis, data mining, scheduling, and VLSI design. The significant SIMT compute power of a GPU makes it an appropriate platform to exploit data parallelism in graph partitioning and accelerate the computation. However, irregular, non-uniform, and data-dependent graph partitioning sub-tasks pose multiple challenges for efficient GPU utilization. Some of these challenges include load imbalance, non-coalesced memory accesses, and warp execution inefficiency. In this paper, we describe an effective and methodological approach to enable multi-level graph partitioning on GPUs. Our solution avoids thread divergence and balances the load over GPU threads by dynamically assigning appropriate number of threads to process the graph vertices and their irregular sized neighbors. Our design is autonomous, i.e., all the steps are carried out by the GPU with minimal CPU involvement, which is required for a range of GPU applications as a pre-processing step. We show that our approach performs better and is comparable in partitioning quality with respect to the state-of-the-art CPU-based parallel graph partitioner (mtmetis). Moreover, to the best of our knowledge, it is the first autonomous approach on GPU.”

Possible Reference 6:
Dash, S. K., et al. “Modular Design of Data-Parallel Graph Algorithms.” *2013 International Conference on High Performance Computing & Simulation (HPCS), High Performance Computing and Simulation (HPCS), 2013 International Conference On*, July 2013, pp. 398–404. *EBSCOhost*, doi:10.1109/HPCSim.2013.6641446.

“Amorphous Data Parallelism has proven to be a suitable vehicle for implementing concurrent graph algorithms effectively on multi-core architectures. In view of the growing complexity of graph algorithms for information analysis, there is a need to facilitate modular design techniques in the context of Amorphous Data Parallelism. In this paper, we investigate what it takes to formulate algorithms possessing Amorphous Data Parallelism in a modular fashion enabling a large degree of code re-use. Using the betweenness centrality algorithm, a widely popular algorithm in the analysis of social networks, we demonstrate that a single optimisation technique can suffice to enable a modular programming style without loosing the efficiency of a tailor-made monolithic implementation.”

Possible Reference 7:

Lin, Jie, et al. “An Optimization and Auto-Tuning Method for Scale-Free Graph Algorithms on SIMD Architectures.” *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), Ubiquitous Computing and Communications (ISPA/IUCC), 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on, ISPA-IUCC*, Dec. 2017, pp. 533–541. *EBSCOhost*, doi:10.1109/ISPA/IUCC.2017.00088.

“The advent of heterogeneous HPC systems, such as the Intel Xeon Phi with many integrated cores and wide SIMD lanes, has been widely used by researchers to accelerate scale-free graph algorithms. However, existing methods are not efficiently considering both the grouping strategy and the selection of the optimizing size for tiling the graph, which would decrease the efficiency of the graph computation. In this paper, we propose a novel Optimization and Auto-Tuning Method on SIMD architecture for scale-free graph algorithms, which is comprised of two sections. Firstly, in the Bucket Grouping Approach, an algorithm-based optimization has been employed to transform the scale-free graph data to conflict-free groups, which can decrease the preprocessing time and improve the SIMD efficiency of scalefree graph applications. Secondly, as optimal tile sizes are changing for different applications of different input graphs, an Auto-Tuning Strategy to optimize tile size achieves effective execution of graph computations. Two typical scale-free graph algorithms, Bellman-Ford and PageRank, are elected to evaluate the performance of OATM. According to the experiment results, OATM performs much better than existing methods, obtaining an average speedup of 1.1×, on these two applications both, by using OATM, comparing to the state-of-the-art methods.”

Possible Reference 8:

Dudas, Adam, et al. “Optimization Design for Parallel Coloring of a Set of Graphs in the High-Performance Computing.” *2019 IEEE 15th International Scientific Conference on Informatics, Informatics, 2019 IEEE 15th International Scientific Conference On*, Nov. 2019, pp. 000011–000018. *EBSCOhost*, doi:10.1109/Informatics47936.2019.9119253.

“This paper presents solution to problem of edge coloring of sizable set of cubic graphs and examination of relations between these graphs. We solved this problem on various computing systems and for various sizes of the problem (various number of graphs). For the computations we used High-Performance Computing Cluster and Amazon Web Services cloud environment. We measured and analyzed time of computation of edge coloring and other properties. Largest set we worked with contained almost 10 million graphs. We created new methodology, which can be used to finding order of the edges which optimizes time of computation of edge coloring for certain subset of graphs. On the basis of this methodology, we implemented algorithm for parallel edge coloring of set of graphs. For testing of the methodology, we designed 8 experiments. Results showed, that worst time of edge coloring of graph from set of 19 935 graphs before use of the methodology was 1260 ms. After application of our methodology, we found same order of edge coloring for whole group of 19 935 graphs and the highest time of coloring was 10 ms.”

Possible Reference 8:
John Hopcroft and Robert Tarjan. 1973. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM* 16, 6 (June 1973), 372–378. DOI:https://doi.org/10.1145/362248.362272

"Efficient algorithms are presented for partitioning a graph into connected components, biconnected components and simple paths. The algorithm for partitioning of a graph into simple paths of iterative and each iteration produces a new path between two vertices already on paths. (The start vertex can be specified dynamically.) If *V* is the number of vertices and *E* is the number of edges, each algorithm requires time and space proportional to max (*V*, *E*) when executed on a random access computer."

Possible Reference 9:

Akif Rehman, Masab Ahmad, and Omer Khan. 2020. Exploring accelerator and parallel graph algorithmic choices for temporal graphs. In *Proceedings of the Eleventh International Workshop on Programming Models and Applications for Multicores and Manycores* (*PMAM '20*). Association for Computing Machinery, New York, NY, USA, Article 7, 1–10. DOI:https://doi.org/10.1145/3380536.3380540

"Many real-world systems utilize graphs that are time-varying in nature, where edges appear and disappear with respect to time. Moreover, the weights of different edges are also a function of time. Various conventional graph algorithms, such as single source shortest path (SSSP) have been developed for time-varying graphs. However, these algorithms are sequential in nature and their parallel counterparts are largely overlooked. On the other hand, parallel algorithms for static graphs are implemented as ordered and unordered variants. Unordered implementations do not enforce local or global order for processing tasks in parallel, but incur redundant task processing to converge their solutions. These implementations expose parallelism at the cost of high redundant work. Relax-ordered implementations maintain local order through per-core priority queues to reduce the amount of redundant work, while exposing parallelism. Finally, strict-ordered implementations achieve the work efficiency of sequential version by enforcing a global order at the expense of high thread synchronizations. These parallel implementations are adopted for temporal graphs to explore the choices that provide optimal performance on different parallel accelerators. This work shows that selecting the optimal parallel implementation extracts geometric performance gain of 46.38% on Intel Xeon-40 core and 20.30% on NVidia GTX-1080 GPU. It is also shown that optimal implementation choices for temporal graphs are not always the same as their respective static graphs."