FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

Assignment

BMCS3003 DISTRIBUTED SYSTEMS AND PARALLEL COMPUTING

2023/2024

Student's name/ ID Number : Hue Zhen Wei / 22WMR05658

Student's name/ ID Number : Lee Wee Harn / 22WMR05673

Student's name/ ID Number : Ricky Hwong Yu Jun / 22WMR05696

Programme : RSW

Tutorial Group : G6

Date of Submission to Tutor : 1st September 2023

Table of Content

Network Analysis: Large-Scale Graph Processing

There is a growing demand to analyze large scale graphs in today's connected society. In network analysis, large-scale graph processing is the complete study of large networks or graphs applying modern computer approaches. These networks are made up of nodes and edges that are linked together to represent entities and their relationships. Conventional analysis tools become insufficient as these networks grow larger and more complicated. As a result, large-scale graph processing is required to obtain significant insights from these advanced structures. This field includes a variety of computational approaches and tools that enable researchers and analysts to explore, evaluate, and extract useful information from large networks *(Amin Sahebi et al., 2023)*.

Large-scale graph processing is critical in recognizing human interactions and behavior in the context of social networks. Researchers may discover influencers, communities, and patterns of information flow by analyzing the connections between individuals. There is a growing demand to analyze large scale graphs in today's connected society. The World Wide Web hyperlink network, for example, includes billions of nodes, Twitter has over 100 million active users and the active Facebook graph has 721 million nodes. Understanding the structure and features of such large graphs, as well as performing algorithms on them, provides an important computing challenge *(Quick, Wilkinson and Hardcastle, 2012)*. Furthermore, Graph-based storage systems are a new method that uses the concepts of large-scale graph processing to change the data management environment. These systems add a new dimension of flexibility and context in storage by showing data as nodes connected by edges. Ownership, version history, and dependencies, for example, can be easily modeled using graph structures, allowing for more complex data organization and retrieval. Graph algorithms effortlessly navigate complex data relationships, optimizing data access patterns. Graph storage systems, like large-scale graph processing techniques, distribute data across nodes to provide scalability and performance. The concept includes query optimization, data lineage tracking, and processing data generated by IoT *(Elyasi, Choi and Sivasubramaniam, 2019)*.

In conclusion, large-scale graph processing provides many advantages across multiple fields by effectively unraveling complicated links inherent in big datasets. Its ability to detect complicated patterns, relationships and hierarchies yields deep insights that typical data analysis approaches frequently ignore. Utilizing the potential of graph processing will likely become even more important in driving improvements in fields such as social networks, recommendation systems, and bioinformatics as technology continues to evolve.

Problem statement

The problem statements are shown as following:

- **Graph applications have lower overall rate of computation and memory access** *(Liu et al., 2020)*
  Heavy computing is prioritized in traditional applications, frequently overshadowing issues with memory access and input/output (I/O) activities *(Liu et al., 2020)*. However, the optimal performance might not be achieved by merely shifting graph processing from memory to a distant memory architecture *(Wang et al., 2023)*. As opposed to this, graph applications concentrate on the interactions between data vertices, where each vertex only takes a little amount of processing but complex interaction. This highlights efficient data processing and memory access latency as crucial performance aspects. Due to the continual data interchange between vertices in graphs, memory access often taken for granted in conventional apps becomes a bottleneck. Graph apps also rely substantially on I/O operations since they handle huge datasets that need for effective loading and storage. This highlights the distinctive nature of graph application performance by shifting the optimisation focus from raw compute to memory access efficiency, data management, and I/O optimisation.

- **Graph applications have complex data dependency** *(Liu et al., 2020)*
  In traditional data processing scenarios, the occurrence of multiple tasks simultaneously accessing the same storage locations is infrequent, allowing for unhindered parallelism due to the predictable and independent nature of data operations *(Liu et al., 2020)*. However, the landscape changes dramatically in the realm of graph applications. These applications center around intricate relationships between vertices and their adjoining edges, leading to a fundamental interdependency between tasks and data elements. Unlike traditional data, graph tasks are intimately tied to the connectivity of vertices, which complicates the prospect of parallelism. The dynamic and often unpredictable nature of vertex interactions makes it challenging to accurately foresee the structure of computations beforehand. Consequently, since it is very difficult to identify general guidelines on the basis of the little experience currently available, the analysis must be undertaken on each individual practical instance under unique assumptions. *(Conti, 2009)*.

- **Graph applications follow unstructured distribution** *(Liu et al., 2020)*

  The graph data structure, renowned for its unstructured and irregular nature, poses unique challenges for parallel processing. Unlike regular data structures that enable straightforward partitioning, the irregularity of graphs complicates efforts to extract parallelism efficiently *(Liu et al., 2020)*. Traditional machine learning techniques can't be used on unstructured data because they call for the construction of a feature matrix with a particular size and arrangement of input samples *(Ogoke et al., 2021)*. This issue is exacerbated by the prevalence of power-law features in many real-world graphs, where a few nodes possess an exceedingly high number of connections, while the majority exhibit much fewer connections. This power-law distribution creates a significant imbalance in the degrees of nodes, thereby aggravating the load imbalance problem and amplifying communication overhead during parallel processing. Scalability can be quite limited by unbalanced computational loads resulting from poorly partitioned data *(Lumsdaine et al., 2007)*. Consequently, addressing these intricacies in graph data becomes crucial for devising effective parallel algorithms and scalable solutions that harness the inherent complexities of real-world graphs.

Literature Review

Pagerank Algorithm

PageRank was named after Larry Page, one of the founders of Google. It is a Google search engine algorithm that ranks a web page with quality keywords and counts links as relations from other websites that refer to the specified web pages. It has an impact on newly introduced websites that will be ranked using adverts and marketing from established websites. The importance or authority of a page is determined by how it is related to other known web pages and how suggested websites are identified by their score *(SEO, 2022)*. The PageRank algorithm generates a probability distribution that represents the possibility that a random user clicking on links will arrive at any given page. PageRank may be determined for any large collection of documents. Several studies assume that the distribution is uniformly distributed among all documents in the collection at the beginning of the computational procedure. PageRank computations involve many "iterations" over the collection to modify estimated PageRank values to more accurately represent the theoretical actual value *(Bisht, 2017)*.

$$pr(u) = \frac{1-d}{n} + d * \sum_{(v,u)\in E} \frac{pr(v)}{q}$$

Figure 1 Formula of PageRank Algorithm

The algorithm's fundamental understanding calculates a page's rank based on its incoming channels. If the site's inbound link comes from a highly ranked page, it is safe to assume that the current page is similarly highly ranked. The rank pr of node u is officially defined as ***Figure 1***. where n represents the number of pages, q is the number of hyperlinks on page v, and d is the damping value, which is set at 0.85. The first component of the equation represents a random probability of arriving at node v from another position. In this scenario, d reflects the possibility of selecting node u from a total set of n nodes. The components provided by all incoming nodes arriving at node v are described in the second section of the equation. The learning factor d is multiplied by the sum of the PageRank values from the arriving nodes of v *(Eedi et al., 2021)*.

In conclusion, to apply the PageRank algorithm to large-scale graph processing, efficient and scalable approaches must be used to compute the value scores of nodes in a big graph. Several ways can be used to accomplish this. To get started, the graph can be divided into smaller subgraphs that can be stored in memory, allowing for parallel processing on several compute nodes. This partitioning can be optimized to maximize computation while minimizing inter-node communication. Furthermore, approaches such as graph trimming and approximate calculations can be used to reduce computational and memory needs while maintaining the quality of the PageRank results. By combining these strategies, it is possible to efficiently apply the PageRank algorithm to large-scale graphs, allowing significant insights to be extracted from big and complicated datasets *(Amin Sahebi et al., 2023)*.

Methodology

OpenMP

OpenMP is a standard parallel programming API written in C, C++, or FORTRAN for shared memory configurations. It consists of a set of "lightweight" compiler directives, library functions, and environment variables that control run-time behavior. The OpenMP Architecture Review Board controls the protocol, which is specified by a number of hardware and software companies. The OpenMP implementation has a direct impact on the OpenMP behavior. This implementation's capabilities allow the programmer to divide the program into serial and parallel regions rather than just continuously running threads, hide stack management, and allow development synchronization. Having stated that, OpenMP cannot guarantee performance, parallelize dependencies, or avoid data racing. The programmer is responsible for data racing, keeping track of dependencies, and aiming toward a speedup *(What is OpenMP, 2012).*

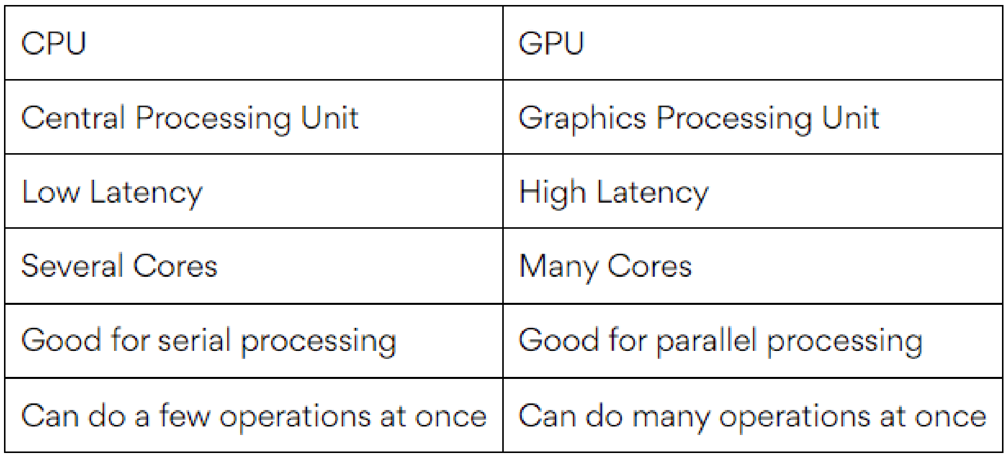
OpenMP is a useful tool in the domain of parallel programming, providing a streamlined approach to utilizing the possibilities of shared-memory multiprocessing settings. OpenMP allows developers to specify parts of code to be executed concurrently by strategically using compiler directives and library calls, resulting in what are known as parallel areas *(curc.readthedocs.io, 2021)*. When a program reaches a parallel zone, it creates a team of threads, each of which executes its own instance of the enclosing code. This is especially useful for activities that can be separated into distinct units of work, because OpenMP maintains data exchange and synchronization among threads automatically. Work-sharing constructs, such as parallel loops, allow for the allocation of jobs, increasing efficiency. Despite its efficiency, OpenMP demands careful consideration of data consistency and synchronization complexities. OpenMP plays a critical role in enhancing the performance of computational applications by enabling the design of parallel programs and providing a mechanism for using the full capability of current multicore processors *(Layton, 2023)*.

To use OpenMP as a PageRank algorithm for large-scale graph processing, we can take advantage of parallelism in the technique's iterative structure. OpenMP is ideal for shared-memory parallelism. Each iteration of PageRank involves adjusting node scores based on their neighbors' scores. This method can be parallelized by allocating different nodes or groups of nodes to different threads. The thread responsible for a node's update can separately calculate the new score using the previous scores of its neighbors during each iteration. However, when updating node scores, we must manage data dependencies and maintain proper synchronization by employing atomic actions or crucial sections. To maximize parallel execution, we can also parallelize additional sections of the algorithm, such as initializing scores, dealing with dangling nodes, and normalizing scores. We may speed up the development of the PageRank algorithm on big graphs by purposefully adding OpenMP directives and optimizing thread management. This improves both performance and scalability.

CUDA

The CUDA (Computer Unified Device Architecture) platform is a software framework developed by NVIDIA to expand the capabilities of GPUs acceleration (*Understanding NVIDIA CUDA: The Basics of GPU Parallel Computing*, 2022). CUDA NVIDIA is able to achieve higher parallelism and efficiency than ordinary CPU using parallel processes. CUDA is a parallel computing platform and programming model developed by NVIDIA corporation *(Heller, 2018)*. At first, CUDA for general purpose in computing its own GPUs as known as graphics processing units, in like matters CUDA enables developers to speed up compute-intensive applications by harnessing the power of GPUs for parallel computing *(Heller, 2018)*. CUDA technology enables parallel processing by breaking down a task into thousands of smaller "threads", hence CUDA technology intensively boosts the performance of NVIDIA GPUs *(Understanding NVIDIA CUDA: The Basics of GPU Parallel Computing, 2022)*. CUDA can be implemented by a wide range of industries or sectors such as computer graphics such as OpenCL language, computational finance, machine learning and even large scale graph processing *(Understanding NVIDIA CUDA: The Basics of GPU Parallel Computing, 2022)*. In short, CUDA enables accelerated computing, as a specialized programming language that runs on GPU CUDA.

NVIDIA CUDA provides a simple C/C++ based interface, again the compiler leverages parallelism built into the CUDA programming model as it compiles the program into code (*Understanding NVIDIA CUDA: The Basics of GPU Parallel Computing*, 2022). Indeed, the CPUS are known to handle a single task simultaneously. In contrast, CUDA GPU is known to handle numerous or even thousands of tasks simultaneously. Not to mention, CUDA GPUs use a parallel computing model, defined as many calculations co-occur instead of executing in sequences (*Understanding NVIDIA CUDA: The Basics of GPU Parallel Computing*, 2022).



CPU	GPU
Central Processing Unit	Graphics Processing Unit
Low Latency	High Latency
Several Cores	Many Cores
Good for serial processing	Good for parallel processing
Can do a few operations at once	Can do many operations at once

Figure 2 shows the differences of CPU and GPU

MPI

Message passing interface (MPI) is a standardized means of exchanging messages between multiple computers running a parallel program across distributed memory (*What Is Message Passing Interface (MPI)? - Definition from WhatIs.com*, 2022). Using MPI allows programs to scale beyond the processors and shared memory of a single computer server, to the distributed memory and processors of multiple computer servers combined together (*Introduction to MPI - Research Computing Services*, 2022). The multiple computers or even multiple processor cores within the same computer in parallel computing are called nodes. Nodes focuses on a particular aspect of the larger computer issue (*What Is Message Passing Interface (MPI)? - Definition from WhatIs.com*, 2022). An MPI communicator may be dynamically established with MPI, allowing numerous processes to operate simultaneously on various cluster nodes. Each process operates independently from the other processes, has a distinct MPI rank to identify it, and has its own memory space. When a programme job is divided into manageable bits and distributed across the processes, each process completes its portion in parallel (*Message Passing Interface :: High Performance Computing*, 2022). MPI defines useful syntax for routines and libraries in programming languages including Fortran, C, C++ and Java (*What Is Message Passing Interface (MPI)? - Definition from WhatIs.com*, 2022).

There are several benefits of the Message passing interface (MPI). First benefit is Portability. MPI has been implemented for many distributed memory architectures, meaning users don't need to modify source code when porting applications over to different platforms that are supported by the MPI standard (*What Is Message Passing Interface (MPI)? - Definition from WhatIs.com*, 2022). The benefit of its widespread use is that customers may easily convert their programmes to the many platforms that the MPI standard supports without having to make significant source code changes. This capability allows for smoother transitions between various hardware configurations and allows developers to concentrate more on optimising their algorithms rather than having to deal with platform-specific complexities. Not only that, speed is also one of the benefits. Vendor implementations might potentially be tuned to take use of built-in hardware characteristics (*What Is Message Passing Interface (MPI)? - Definition from WhatIs.com*, 2022). MPI implementations are often finely tuned to capitalize on the specific characteristics of the hardware they operate on, ensuring efficient communication and computation within distributed memory architectures. Furthermore, vendor-specific MPI implementations may exploit native hardware features, maximizing the speed and resource utilization of parallel applications.

In the past, MPI was a widely used platform for the parallel calculation of the PageRank technique, but this required users to pay too much attention to each operation they performed, such as load balancing and data layout. In a single iteration, calculating PageRank for a given collection of sites is not difficult; however, because the usual web network is so big, parallelization is frequently necessary *(Forno et al., 2021)*. The configuration and PageRank loop are the two key processes that make up the calculation. In the configuration process, the master MPI worker reads the graph edges from a file and broadcasts them to all workers *(Forno et al., 2021)*. Each worker analyses the allocated subgraph when all workers have received the graph data, identifying the incoming and outgoing connections for every node *(Forno et al., 2021)*. For the purpose of determining the PageRank values for each node, certain incoming and outgoing link counts are required *(Forno et al., 2021)*. Next is the PageRank loop. PageRank values are repeatedly updated by workers using their neighbours' rankings and incoming links *(Forno et al., 2021)*. Following the PageRank loop, employees use MPI communication to exchange their newly updated PageRank values *(Forno et al., 2021)*. Up until the PageRank numbers stabilize, signalling convergence, the process iterates continuously *(Forno et al., 2021)*. In order to indicate the relevance of each node in the graph, each worker generates PageRank values for the nodes in its subgraph *(Forno et al., 2021)*.

Reference

1. Amin Sahebi, Barbone, M., Procaccini, M., Luk, W., Georgi Gaydadjiev, & Giorgi, R. (2023). Distributed large-scale graph processing on FPGAs. Journal of Big Data, 10(1). https://doi.org/10.1186/s40537-023-00756-x

2. Quick, L., Wilkinson, P., & Hardcastle, D. (2012, August 1). Using Pregel-like Large Scale Graph Processing Frameworks for Social Network Analysis. IEEE Xplore. https://doi.org/10.1109/ASONAM.2012.254

3. Elyasi, N., Choi, C., & Sivasubramaniam, A. (2019). Large-Scale Graph Processing on Emerging Storage Devices This paper is included in the Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST '19). Large-Scale Graph Processing on Emerging Storage Devices. https://www.usenix.org/system/files/fast19-elyasi.pdf

4. What is OpenMP? (2021). Www.tutorialspoint.com. https://www.tutorialspoint.com/what-is-openmp

5. Using OpenMP with C — Research Computing University of Colorado Boulder documentation. (2021). Curc.readthedocs.io. https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html

6. Layton, J. (2023). Introduction to OpenMP» ADMIN Magazine. ADMIN Magazine. https://www.admin-magazine.com/HPC/Articles/Parallel-Programming-with-OpenMP

7. SEO, H. (2022, December 5). What is PageRank? PageRank Algorithm Definition and Analysis - Holistic SEO. Www.holistic seo.digital. https://www.holisticseo.digital/theoretical-seo/pagerank/

8. Bisht, J. (2017, August 30). Page Rank Algorithm and Implementation - GeeksforGeeks. GeeksforGeeks. https://www.geeksforgeeks.org/page-rank-algorithm-implementation/

9. Eedi, H., Peri, S., Ranabothu, N., & Utkoor, R. (2021, March 1). An Efficient Practical Non-Blocking PageRank Algorithm for Large Scale Graphs. IEEE Xplore. https://doi.org/10.1109/PDP52278.2021.00015

10. *Understanding NVIDIA CUDA: The Basics of GPU Parallel Computing*. (2022, December 29). Www.turing.com. https://www.turing.com/kb/understanding-nvidia-cuda

11. Heller, M. (2018, August 30). *What is CUDA? Parallel programming for GPUs*. InfoWorld. https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.htm l

12. *What is message passing interface (MPI)? - Definition from WhatIs.com*. (2022, April 23). SearchEnterpriseDesktop. https://www.techtarget.com/searchenterprisedesktop/definition/message-passing-interface-MPI

13. *Introduction to MPI - Research Computing Services*. (2022, September 24). Carleton.ca. https://carleton.ca/rcs/rcdc/introduction-to-mpi/#:~:text=The%20Message%20Passing%20Interface%20

14. *Message Passing Interface :: High Performance Computing*. (2022, April 18). Hpc.nmsu.edu. https://hpc.nmsu.edu/discovery/mpi/introduction/

15. Liu, N., Li, D., Zhang, Y., & Li, X. (2020). Large-scale graph processing systems: a survey. 21(3), 384–404. https://doi.org/10.1631/fitee.1900127

16. Conti, S. (2009). Analysis of distribution network protection issues in presence of dispersed generation. Electric Power Systems Research, 79(1), 49–56. https://doi.org/10.1016/j.epsr.2008.05.002

17. Wang, J., Li, C., Liu, Y., Wang, T., Mei, J., Zhang, L., Wang, P., & Guo, M. (2023). Fargraph+: Excavating the parallelism of graph processing workload on RDMA-based far memory system. Journal of Parallel and Distributed Computing, 177, 144–159. https://doi.org/10.1016/j.jpdc.2023.02.015

18. Ogoke, F., Meidani, K., Hashemi, A., & Farimani, A. B. (2021). Graph convolutional networks applied to unstructured flow field data. Machine Learning: Science and Technology, 2(4), 045020. https://doi.org/10.1088/2632-2153/ac1fc9

19. Lumsdaine, A., Gregor, D., Hendrickson, B., & Berry, J. W. (2007). CHALLENGES IN PARALLEL GRAPH PROCESSING. Parallel Processing Letters, 17(01), 5–20. https://doi.org/10.1142/s0129626407002843

20. Forno, E., Salvato, A., Macii, E., & Urgese, G. (2021, May 28). PageRank Implemented with the MPI Paradigm Running on a Many-Core Neuromorphic Platform. MDPI. https://doi.org/10.3390/jlpea11020025

Appendix 2 – Assessment Rubric (Part A) 40% – CLO2

Name(s): Hue Zhen Wei, Lee Wee Harn, Ricky Hwong Yu Jun   Programme: RSW   Group: G6   Date: 01/09/2023

Criteria	Weak	Average	Good	Excellent	Mark
Introduction (30)	• No or very little discussion on an existing problem and the project The proposed project already exists, or with very minor change. • No discussion or very little introduction given to the related system or technology. 0 - 9	• Little discussion on an existing problem and introduction of a proposed project. • Minor ideas are modified from existing system(s). • Introduction to the related system is given, but no evaluation provided. 10 - 18	• Good discussion and evaluation of an existing problem and the proposed project. • Ideas modified from an existing system, with some creative ideas are added. • Good discussion and evaluation of the related system. 19 - 24	• A very good discussion and evaluation of an existing problem and the proposed project. • Majority of the ideas are creative. • A very good discussion and evaluation of the related system. 25 - 30	
Literature Review (30)	• Contents are retrieved directly from the literature without any paraphrasing. • Selected literature was from unreliable sources (e.g. Web sources). No critical evaluation was provided. 0 - 9	• Summary is lengthy, contents are retrieved directly from the literature without any critical evaluation. • Selected literature was from unreliable sources. Literary supports were vague or ambiguous. 10 - 18	• Summary is concise and clear. Able to identify key constructs and variables related to the research questions, from the relevant, reliable theoretical and research literature. • Discussions on the validity, opinions, arguments and evidence are presented. 19 - 24	• Summary is concise and clear, which integrates critical and logical details from the peer- reviewed theoretical and research literature. • Attention is given to different perspectives, conditionality, threats to validity, and opinion vs. evidence. 25 - 30	
Methodology (30)	• No discussion or very little introduction given to the methods applied. • Brief design is provided but lack of explanation. • Algorithm suggested is not appropriate or less relevant. 0 - 9	• Brief description of system design, with some explanations. • Introduction to the related application of the methods is given, but no evaluation provided. • Algorithm suggested is relevant, but lack of understanding or explanation. 10 - 18	• System design is well-illustrated, and with clear explanation. • Good discussion and evaluation of the methods applied. • Algorithm suggested is relevant and a good explanation shows understanding. 19 - 24	• System design is well-illustrated, with good explanation. • Good discussion and evaluation of the relevant and practical methods applied. • Algorithm suggested is relevant and good explanations to relate to the proposed project. 25 - 30	
Reference (10)	• No proper referencing is done. Only rely on website content, but no research papers. Reference list is not provided. 0 - 2	• Reference list is provided but not with APA Referencing standard. Some mixture of reference sources. 3 - 5	• Referencing is done properly. Some mixture of APA standard reference sources with at least one journal. 6 - 8	• Rich mixture of reference sources especially good quality of research papers. Proper citations are done whenever necessary. Reference list follows the proper APA Referencing standard and the information is complete. 9 - 10	
				Sum of scores	
				Final score = sum of scores/100*40 (base 40%)	