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 : 22nd September 2023

Table of content

Title: Network Analysis - Large Scale Graph Processing

1.0 Abstract

Large-scale graph analysis is a critical element of any big data analysis toolkit, as it plays a pivotal role in modelling and dissecting complex social and information networks. However, handling graphs containing hundreds of billions of edges necessitates the use of distributed graph mining systems and distributed algorithms *(Home, 2020)*. One such influential algorithm is PageRank, famously employed by Google Search to rank websites in search engine results. Named after Larry Page, one of Google's co-founders, PageRank is a technique for gauging the significance of web pages within the vast online landscape, making it an indispensable tool for navigating and understanding the digital ecosystem *(Bisht, 2017)*.

A dataset based on Amazon's network that contains 1234877 rows of data is used in this research to calculate the top 10 nodes by PageRank. This analysis gives useful insights on Amazon's most prominent nodes, contributing to a better understanding of the dynamics and value of many components in this huge online retail ecosystem. Based on the number of incoming relationships and the significance of the related source nodes, the PageRank algorithm determines the importance of each node in the network. In general, the idea that underlies it is that a page is only as important as the pages that link to it *(PageRank - Neo4j Graph Data Science, 2021)*. Therefore, the PageRank algorithm provides a quantitative measure of the influence and centrality of individual nodes in the Amazon network, shedding light on which products, users, or entities play pivotal roles within the vast digital marketplace.

In addition to the PageRank investigation on Amazon's network, this research examines the impact of advanced parallel computing techniques such as OpenMP, MPI and CUDA on execution time and performance. To be more specific, OpenMP is a set of compiler directives as well as an API for programs written in C, C++, or FORTRAN that provides support for parallel programming in shared-memory environments *(What Is OpenMP?, 2019)*. On the other hand, MPI is a standardised means of exchanging messages between multiple computers running a parallel program across distributed memory *(What Is Message Passing Interface (MPI), 2020)*. Finally, CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on its own GPUs *(Heller, 2018)*.

Incorporating these advanced parallel computing techniques into our analysis is crucial for evaluating their potential to optimise the performance of the PageRank algorithm on Amazon's network, particularly in terms of execution time. Each of these technologies brings unique advantages and capabilities to the

table in the context of reducing execution time. OpenMP, for instance, plays a vital role in enhancing the efficiency of computations on shared-memory systems. By parallelizing tasks and distributing them among multiple processor cores, OpenMP can significantly reduce execution time for compute-intensive tasks. This means that when processing the vast amount of data within Amazon's network, we can expect to see substantial improvements in execution speed, allowing us to obtain results more quickly. MPI, on the other hand, addresses the challenge of communication and coordination in distributed environments. Efficient message passing facilitated by MPI can lead to faster data exchange between nodes, which is essential for parallel processing. This reduction in communication overhead directly contributes to shorter execution times, particularly in scenarios where data needs to be shared and processed across various locations within Amazon's network. Furthermore, CUDA introduces a different dimension by harnessing the computational power of GPUs. With their parallel architecture, GPUs can perform a multitude of calculations simultaneously, which can drastically reduce the execution time of complex algorithms like PageRank. When CUDA is applied to the computation-intensive aspects of our analysis, we can expect a remarkable acceleration in execution speed, resulting in quicker insights and more responsive network analysis.

In conclusion, through this investigation, we aim to determine not only the most suitable parallel computing approach but also the optimal combination of these techniques to expedite the processing of the extensive data involved in our network analysis. Ultimately, our goal is to achieve substantial reductions in execution time, allowing us to obtain more efficient and accurate results in a timelier manner, which is critical for making informed decisions within the dynamic and rapidly evolving environment of Amazon's network.

2.0 Result

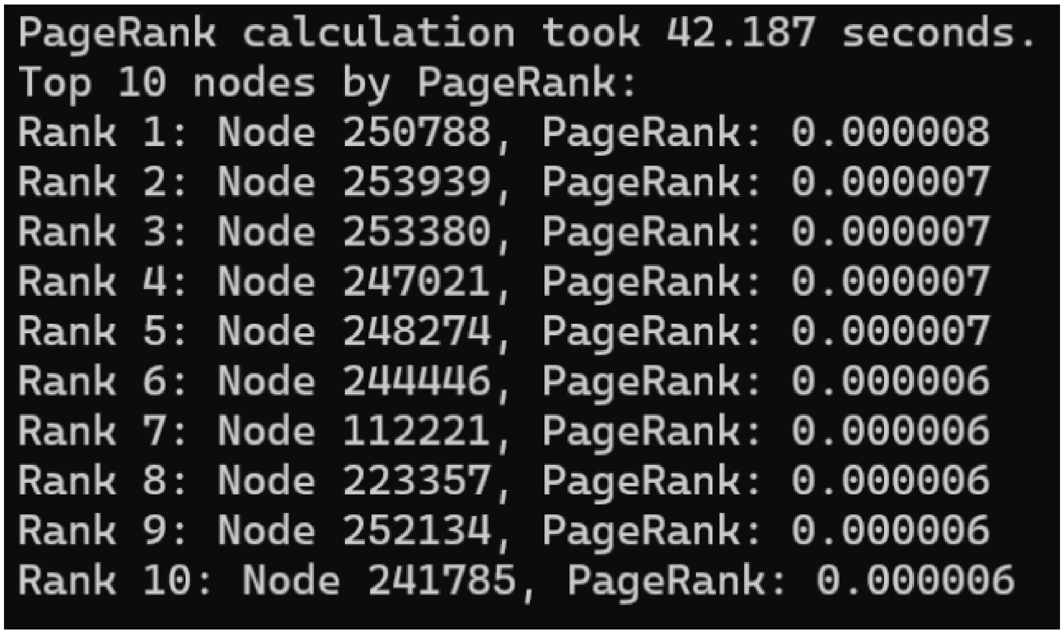
The hardware requirements of a computer system have a significant influence on code execution and parallel and distributed system parallel computing. The computational capacity and speed at which code can operate are determined by hardware components such as the operating system, processor, memory, graphics processing unit and storage. A strong hardware configuration with more processor cores and enough memory can perform more complicated and computationally difficult jobs, allowing for quicker code execution. The capacity of hardware to handle parallelism is critical in the context of parallel and distributed computing. Parallel processing needs more cores or processors to operate on separate sections of the code at the same time, considerably speeding up job execution and file reading. Each of which contributes processing power to form desired output results by performing analysis. As a result, hardware infrastructure is crucial in determining the efficiency, performance and scalability of distributed code implementations. In summary, a system's hardware requirements shape how code executes and how efficiently parallel and distributed computing may be used to optimise performance and accomplish desired outcomes.

Visual Studio has a comprehensive collection of tools for debugging code and enabling parallel distribution, building code and publishing applications. Its integrated debugger includes capabilities like as breakpoints, watch windows, and real-time code inspection, making it easier for developers to find and handle errors in their code. Additionally, Visual Studio support the implementation of parallel computing in coding such as openMP, MPI and CUDA. Lightweight but powerful debugging and parallel distribution tools in this IDE speeds up the development process and improves the efficiency of constructing scalable and high-performance software solutions (*Visual Studio IDE, Code Editor, Azure DevOps, & App Center*, 2018).

In this implementation was performed using a laptop provided with hardware requirements as the followed table:

Operating System	Windows 10 Home - ASUS recommends Windows 11
Processors	AMD Ryzen™ 5 4600H Mobile Processor (6-core/12-thread, 11MB Cache, 4.0 GHz max boost)
Memory (RAM)	8GB DDR4-3200 SO-DIMM X 2, Max Capacity:16GB
Storage (ROM)	512GB PCIe® 3.0 NVMe™ M.2 SSD
Graphics	NVIDIA® GeForce® GTX 1650 Ti, 4GB GDDR6

2.1 Result when applying PageRank code as serial



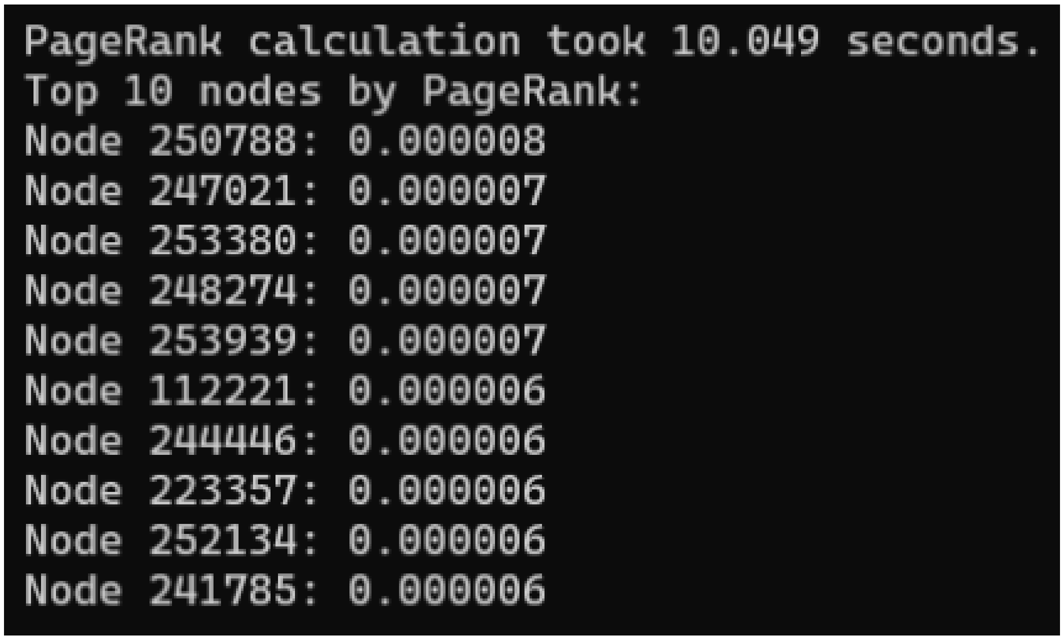
```
PageRank calculation took 42.187 seconds.
Top 10 nodes by PageRank:
Rank 1: Node 250788, PageRank: 0.000008
Rank 2: Node 253939, PageRank: 0.000007
Rank 3: Node 253380, PageRank: 0.000007
Rank 4: Node 247021, PageRank: 0.000007
Rank 5: Node 248274, PageRank: 0.000007
Rank 6: Node 244446, PageRank: 0.000006
Rank 7: Node 112221, PageRank: 0.000006
Rank 8: Node 223357, PageRank: 0.000006
Rank 9: Node 252134, PageRank: 0.000006
Rank 10: Node 241785, PageRank: 0.000006
```

Figure 2.1.1

Based on ***Figure 2.1.1***, this is a result of implementation of the PageRank algorithm without the application of any parallelization methods. It computes PageRank scores for the “Amazon” graph dataset and identifies the top-ranked nodes based on those scores. However, it does not incorporate parallelism to improve computation speed, which may be necessary for processing large-scale graphs efficiently. Therefore, as ***Figure 2.1.1*** shown above, the PageRank calculator took 42.187 seconds to compute the “Amazon” graph dataset that contains 1234877 rows of data.

It begins by reading network data from a file, which is “amazon0302.txt” , constructing an adjacency list representation of the graph, and initialising PageRank scores for each node. The core PageRank calculation loop iteratively updates the scores based on the probability of transitioning from one node to another, factoring in the damping factor and the contributions from neighbouring nodes. Convergence is checked at each iteration using a defined threshold, ensuring that the PageRank scores stabilise. Once the algorithm converges, the code sorts and displays the top-ranked nodes, offering valuable insights into the nodes' significance within the Amazon network. This process leverages the PageRank algorithm to identify nodes of importance, contributing to a better understanding of the network's structure and influential entities.

2.2 Result when applying OpenMP



```
PageRank calculation took 10.049 seconds.
Top 10 nodes by PageRank:
Node 250788: 0.000008
Node 247021: 0.000007
Node 253380: 0.000007
Node 248274: 0.000007
Node 253939: 0.000007
Node 112221: 0.000006
Node 244446: 0.000006
Node 223357: 0.000006
Node 252134: 0.000006
Node 241785: 0.000006
```

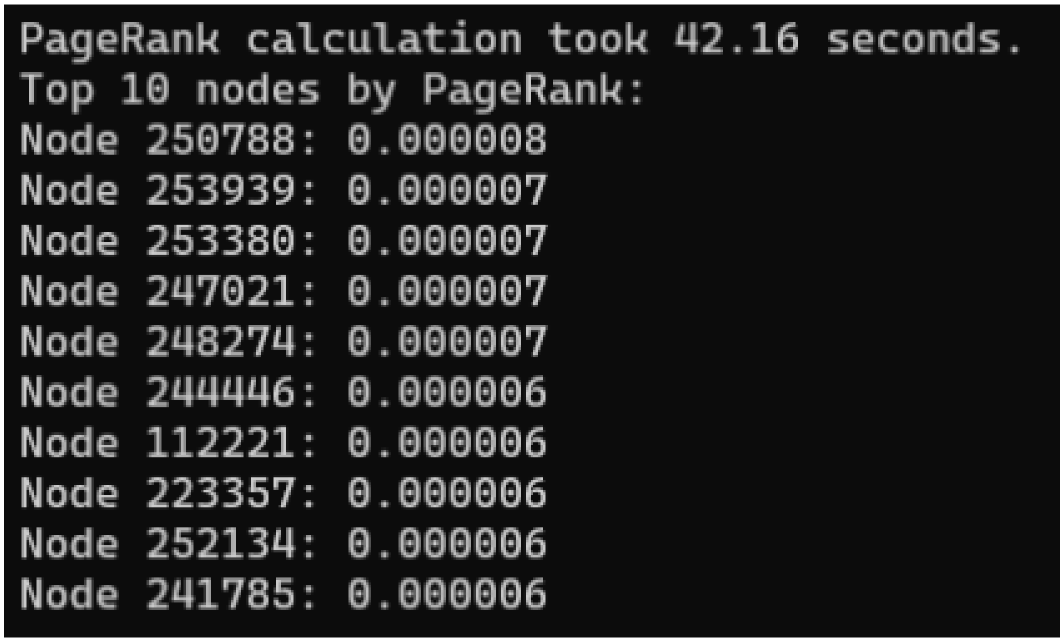
Figure 2.2.1

Based on ***Figure 2.2.1***, OpenMP has been applied to parallelize the computation of PageRank scores for the large-scale graph dataset of Amazon Network. This parallelization aims to take advantage of multiple CPU cores or threads to significantly improve the performance of the PageRank calculation. The result of the top 10 nodes by the PageRank algorithm is the same as the one that doesn't apply any parallel method indicating that with the use of OpenMP, correct results are also being calculated successfully. Besides that, the PageRank calculations are being enhanced from 42.187 seconds to 10.049 seconds. This is because parallelization of OpenMP significantly improves the performance of PageRank calculations, especially on multi-core processors. It allows for faster convergence and reduced execution time, making it feasible to compute PageRank on large graphs efficiently in computing the graph dataset of Amazon Network.

Initially, it reads network data from the "amazon0302.txt", constructs an adjacency list representation of the graph, and initialises PageRank scores for each node. The core of the PageRank calculation is now parallelized using OpenMP directives, allowing multiple threads to concurrently compute PageRank scores for different nodes. This parallelization leads to a substantial reduction in computation time, particularly for large graphs. Convergence is checked as before, and once the algorithm stabilises, the code filters out nodes with infinite PageRank scores and sorts the remaining nodes by their PageRank values. Finally, it displays the top-ranked nodes, providing valuable insights into the most influential entities within the Amazon network. Additionally, the code dynamically sets the number of threads to the maximum available on the system, optimising resource utilisation for the computation.

In addition to the substantial performance improvement achieved through OpenMP parallelization, it's important to note that this enhancement also makes the PageRank algorithm more scalable. With the ability to efficiently compute PageRank scores for large graphs like the Amazon Network dataset, the code becomes more versatile and applicable to various network analysis tasks. Moreover, the dynamic adjustment of the number of threads to the maximum available resources ensures optimal utilisation of the underlying hardware, making it adaptable to different computing environments. This combination of accuracy, speed, and scalability underscores the significance of employing parallel computing techniques like OpenMP in tackling complex graph analysis problems, especially when dealing with vast and intricate networks like the one represented by the Amazon dataset.

2.3 Result when applying MPI



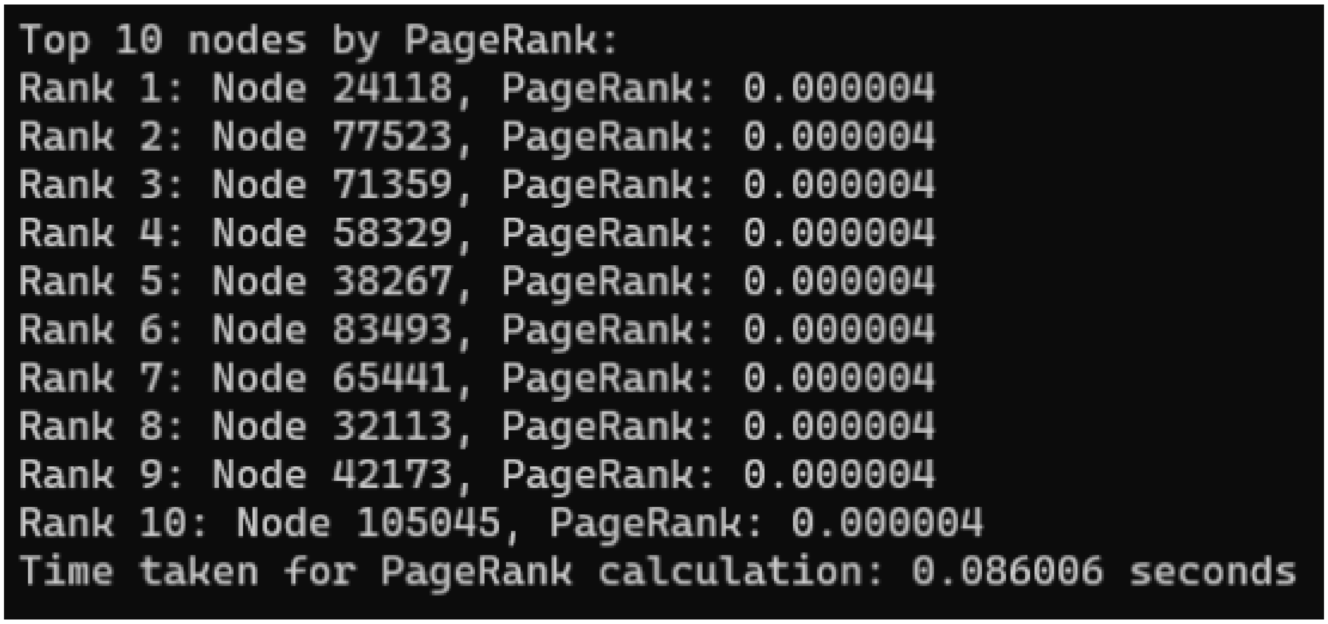
```
PageRank calculation took 42.16 seconds.
Top 10 nodes by PageRank:
Node 250788: 0.000008
Node 253939: 0.000007
Node 253380: 0.000007
Node 247021: 0.000007
Node 248274: 0.000007
Node 244446: 0.000006
Node 112221: 0.000006
Node 223357: 0.000006
Node 252134: 0.000006
Node 241785: 0.000006
```

Figure 2.3.1

Based on the ***Figure 2.3.1*** shows the results of PageRank scores for the large-scale graph dataset of Amazon Network. It loads graph data from a specified file, constructs the graph structure in memory, and performs iterative PageRank calculations, checking for convergence based on a defined threshold. Once the calculation converges, it gathers the results, sorts the nodes by PageRank, and displays the top 10 nodes with the highest scores and also execution time of finding the top 10 nodes. Based on ***Figure 2.3.1*** and ***Figure 2.3.2***, MPI and OpenMP have the same top 10 nodes with the highest scores. But, the execution time for MPI and serial is similar due to some reasons. Similar performance can also be caused by load balancing problems, where some MPI processes may finish their task rapidly and wait for others. To achieve significant speedup compared to a serial technique, efficient parallelization frequently necessitates careful consideration of graph size, load distribution, and communication optimization.

It begins by initialising MPI processes and reading a graph dataset (in this case, "amazon0302.txt") where nodes and their connections are stored. The graph data is read by the root process (rank 0) and distributed among all MPI processes. The calculatePageRank function then computes PageRank scores for each node in the graph, iteratively updating the scores until convergence, using the specified damping factor and convergence threshold. The PageRank calculation is performed concurrently across MPI processes, exploiting parallelism. After computation, the top-ranked nodes are sorted based on their PageRank scores, and the results are displayed, showing the highest-ranking nodes. The code also includes timing measurements to report the execution time of the PageRank calculation. Finally, MPI is finalised to gracefully exit the parallel environment. This code leverages MPI's parallel processing capabilities to efficiently compute PageRank scores for large-scale graphs across multiple distributed computing nodes.

2.4 Result when applying CUDA



```
Top 10 nodes by PageRank:
Rank 1: Node 24118, PageRank: 0.000004
Rank 2: Node 77523, PageRank: 0.000004
Rank 3: Node 71359, PageRank: 0.000004
Rank 4: Node 58329, PageRank: 0.000004
Rank 5: Node 38267, PageRank: 0.000004
Rank 6: Node 83493, PageRank: 0.000004
Rank 7: Node 65441, PageRank: 0.000004
Rank 8: Node 32113, PageRank: 0.000004
Rank 9: Node 42173, PageRank: 0.000004
Rank 10: Node 105045, PageRank: 0.000004
Time taken for PageRank calculation: 0.086006 seconds
```

Figure 2.4.1

Based on ***Figure 2.4.1*** shows that the results of PageRank algorithm implementing CUDA scores are not expected as the serial code from ***Figure 2.1.1.*** CUDA firstly initialises PageRank such as epsilon and damping factor value to perform a series of activities such as calculating PageRank using kernel and convergence function.

In this parallel implementation, <thrust> is one of the parallel algorithms libraries which resembles C++ Standard Template Library (STL) (*Thrust - Parallel Algorithms Library*, 2013). "Thrust" is capable of interoperability with established technology including CUDA and OpenMP which facilitates integration with existing source code. In this case, in order to achieve high performance on calculation from the calculation compared to the PageRank serial code.

Unfortunately, the ***Figure 2.4.1*** expected ranked nodes are not exactly the same from the PageRank algorithm from ***Figure 2.1.1.*** Hence, CUDA implementation in PageRank may be not suitable or factors such as lack of unit testing, debugging or lack of CUDA practices in order to achieve the same or consistent implementation from each other. There are also some technical aspects which result in differing output from ranked nodes including the threshold difference between serial code and the logic from CUDA parsing the value in the function. Since it involves tiny decimals from calculation, it may cause improper synchronization that lead to race conditions and incorrect results. To give an illustration, **Damping Factor** and **Epsilon** are entitled as critical parameters that gradually influence the algorithm's behaviour and convergence. These parameters are expected to be parsed same with serial and CUDA implementation; perhaps it leads to differing results.

In contrast, the results from CUDA may be advantageous in finding these nodes but in different logic, perhaps CUDA lies in the ability to execute reading and perform calculation within 1 seconds which is 0.086006s, when using CUDA is implementing faster convergence and computation, due to parallelism, which is gradually faster than PageRank series output.

3.0 Discussion

3.1 OpenMP

Parallelizing the PageRank calculation with OpenMP provides scalability as well as real-time analysis for large-scale graphs like the Amazon Network. The benefits of parallelism become clearer as the scale of the graph increases. Without parallelism, the PageRank algorithm could become insufficient due to long execution times, making large datasets difficult to handle efficiently. Parallelization allows for the use of many CPU cores, utilising the full computing capacity of current CPUs. This not only improves efficiency but also ensures that the method scales effectively with the size of the graph, allowing it to analyse even the most complex networks *(What Are the Pros and Cons of Using OpenMP, 2020)*. Furthermore, parallel PageRank calculation enables near real-time analysis, which is critical for applications that require quick insights. Parallelism ensures that computations execute concurrently, speeding up the analysis process in scenarios such as monitoring social networks or analysing web page connectivity, when speedy responses are critical. This real-time functionality allows users to keep up with dynamic network changes and respond quickly to emerging patterns or anomalies, improving the PageRank algorithm's overall utility in dynamic and time-sensitive scenarios.

However there are still some limitations when using OpenMP to compute PageRank scores on huge graph datasets like the Amazon network. When certain nodes in the graph have much more connections or neighbours than others, load imbalance occurs. Because of this imbalance, certain processing threads may take significantly longer to finish their jobs, possibly reducing the benefits of parallelism. Furthermore, PageRank calculations require maintaining two critical data structures which are the 'pagerank' and 'new_pagerank' maps. They record the PageRank scores for each node. These data structures can require a significant amount of memory for large-scale graphs like those observed in networks like Amazon's. Furthermore, due to the sensitivity of hyperparameters such as the damping factor and convergence threshold, parameter tuning in the context of PageRank computation using OpenMP is a difficult operation. These hyperparameters have a direct impact on the algorithm's convergence speed as well as the quality of the PageRank results. Obtaining the ideal parameter values frequently necessitates complicated study and experimentation on a wide range of graph configurations, as what works well for one dataset may not work well for another. This iterative process of fine-tuning hyperparameters adds complexity and processing complexity to the implementation, highlighting the significance of strong parameter selection strategies for accurate and efficient PageRank calculations *(Limitations, 2018)*.

To mitigate load imbalance, dynamic load balancing techniques can be implemented, dynamically distributing tasks among threads based on the actual workload of each node. This approach can help ensure that computational resources are utilised more efficiently across all threads, minimising the impact of highly connected nodes on overall performance. Furthermore, to address memory usage concerns, memory-efficient data structures and algorithms can be explored. Sparse data structures, which store PageRank scores only for nodes with non-zero values, can significantly reduce memory requirements. Additionally, employing techniques like graph partitioning and distributed computing frameworks can help distribute the memory load across multiple machines or nodes when dealing with extremely large graphs that cannot fit into a single machine's memory. Finally, regarding hyperparameter tuning, automated techniques such as grid search or Bayesian optimization can be employed to systematically explore the hyperparameter space and find optimal values for different graph structures. This approach can save valuable time and resources while ensuring that the PageRank algorithm is fine-tuned for specific datasets without the need for extensive manual experimentation.

3.2 MPI

The source code presents a parallelized implementation of the PageRank algorithm using MPI for distributed computing. It effectively distributes the task across MPI processes, each of which is in charge of calculating PageRank values for a portion of the graph's nodes. Using MPI functions, communication between processes is effectively controlled, maintaining synchronisation during the iterative PageRank computation. The code starts up PageRank scores, looks for convergence, and uses a broadcast technique to keep processes in sync. Although this method offers a strong foundation for distributed PageRank computations, it is more tolerant of higher communication latencies *(Message Passing Model of Process Communication, 2021)*. It's also important to note that robustness and error handling could be improved for usage in production. The algorithm also gathers and ranks the top nodes according to PageRank, giving information on significant network components. Generally speaking, this code is an effective tool for network analysis jobs, especially when working with large and complicated graph data.

The disadvantages of using MPI for parallelized PageRank include the possibility of high memory consumption due to the use of standard data structures, increased communication overhead between MPI processes, the possibility of load imbalance among processes, reliance on a spartan convergence criterion, restricted error handling, and scaling issues for very large graphs. However, the optimal performance might not be achieved by merely shifting graph processing from memory to a distant memory architecture *(Wang et al., 2023).* It also faces increased communication overhead between MPI processes, potentially leading to performance bottlenecks. Additionally, there's a risk of load imbalance among processes, which can result in inefficient resource utilisation. The code relies on a simplistic convergence criterion and lacks comprehensive error handling, which could affect its accuracy and robustness. Finally, scalability concerns arise when dealing with very large graphs, necessitating further optimization to handle such datasets effectively.

A diverse strategy to improve can handle the inherent difficulties in the field of MPI-parallelized PageRank calculation. First and foremost, it is crucial to address the issue of excessive memory usage, especially when working with vast networks including millions of nodes. Adoption of memory-efficient data structures, such as compressed adjacency matrix representations, is the answer. Asynchronous communication methods, which enable parallel processing and data transmission to reduce communication overhead, are also crucial for improving overall efficiency. It becomes essential to reduce unnecessary data transfers, optimise data partitioning schemes, and use techniques like message aggregation in order to further simplify the process. load balancing methods are used to guarantee fair allocation of computational workloads across processes and prevent the dangers of load imbalance. The method's efficacy can be increased by using more complex convergence criteria, dynamic threshold adaptation based on convergence progress, or early stopping techniques that speed up computing without compromising accuracy. These combined improvements open the door for a parallelized PageRank calculation that is more effective and scalable inside the MPI environment.

3.3 CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform by NVIDIA. It allows developers to use NVIDIA GPUs for general-purpose computing. There are some advantages and disadvantages of using CUDA as parallel computing in the large-scale graphics processing in network analysis. CUDA offers several advantages by allowing simultaneous execution of thousands of threads on GPU cores, which is extremely useful for graph algorithms with many linked nodes and edges. When compared to standard CPU-based techniques, this leads to substantial speedup. Implementing CUDA as one of the solutions, CUDA provides a general-purpose programming language based on C, making it accessible to non-GPU programmers (*What Is CUDA Like? What Is It For? What Are the Benefits? And How to Start?*, 2014).

Besides, the parallel processing capabilities of CUDA result in significant performance increases in graph processing applications, lowering the time needed to analyse and manage large-scale graph data and even perform some useful analysis or produce desired analysis output. CUDA enables developers to leverage the power of GPUs, which are capable of running a large number of operations simultaneously. This can result in significant performance improvements for highly parallelizable problems (*What Is CUDA Like? What Is It For? What Are the Benefits? And How to Start?*, 2014). Aside from performance and efficiency, CUDA is portable to install in different environments as CUDA installation comes with a software development kit (SDK) that includes libraries, debugging tools, profiling tools, and more (*What Is CUDA Like? What Is It For? What Are the Benefits? And How to Start?*, 2014). Therefore, CUDA can be easily integrated with existing software tools and libraries, making it convenient for developers (Frąckiewicz, 2023).

CUDA mainly developed for research instead of normal programmers or juniors (Schaeufele, 2009)l . Speeds and execution or processing time taken might be bottlenecked at the bus between CPU and GPU (Schaeufele, 2009). The major disadvantages of CUDA may produce the variants result based on several considerations. From the code design during the development phase, PageRank algorithm itself computations need floating-point arithmetic, and the precision of floating-point operations varies slightly across CPUs and GPUs. This can lead to extremely tiny numerical variations in the final rankings, but these discrepancies are typically insignificant in practice. Therefore, leading the nodes ranks results met different occurrences, perhaps the CUDA yield significantly quicker results than the CPU-based implementation, especially for big graphs. Lastly, regardless of the PageRank algorithm, logic as the core must remain consistent with CUDA, since CUDA uses Kernel and memory to store data. The implementation parts such as initialization, kernel jobs assigned, memory allocation and lastly convergence checking. Lastly, key management on critical parameters such as **Epsilon** and **Damping**

Factors, both aspects are the key to be considered in order to allow the CUDA implementation and analysis calculations performed to archive same ranked nodes results as serial code PageRank.

From the lessons debugging the CUDA file, it is vital to implement precise management techniques from modifying floating-point precision in both CPU and CUDA versions. This ensures that the tiny numerical discrepancies between the two implementations stay below acceptable limits. Verification processes should be implemented to ensure that the CUDA and CPU implementations give consistent results. This may involve comparing the output ranks and confirming that they match within the stated tolerance levels. Besides, checking the data structures and memory layouts utilised in both implementations are the same. This can assist to avoid any variations in data storing and vector reading out of bounds. To avoid memory used up by destructing host data structures during graph data processing after reading. Last but not least, creating a robust set of unit tests and validation processes to ensure that both implementations are proper and consistent. Debugging and running these tests on a regular basis as part of the development and optimisation process. Then, analyse the performance of both implementations to discover bottlenecks and areas for improvement. Ascertain that performance enhancements in the CUDA version do not compromise result consistency.

4.0 References

1. Home. (2020). Sites.google.com. Retrieved September 20, 2023, from https://sites.google.com/view/lsga/home

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

3. PageRank - Neo4j Graph Data Science. (2021). Neo4j Graph Data Platform. https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/#:~:text=The%20PageRank%20algorithm%20measures%20the

4. What is OpenMP? (2019). Www.tutorialspoint.com. https://www.tutorialspoint.com/what-is-openmp

5. What is message passing interface (MPI). (2020). SearchEnterpriseDesktop.https://www.techtarget.com/searchenterprisedesktop/definition/message-passing-interface-MPI#:~:text=The%20message%20passing%20interface%20(MPI)%20is%20a%20standardized%20means%20of

6. 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.html

7. 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

8. Message Passing Model of Process Communication. (2021, March 1). GeeksforGeeks. https://www.geeksforgeeks.org/message-passing-model-of-process-communication/

9. What are the pros and cons of using OpenMP. (2020). Www.linkedin.com. https://www.linkedin.com/advice/0/what-pros-cons-using-openmp-vs-mpi-shared#:~:text=One%20of%20the%20main%20benefits

10. Limitations. (2018, April 24). Www.ibm.com. https://www.ibm.com/docs/en/xl-c-and-cpp-linux/16.1.0?topic=gpus-limitations

11. *What is CUDA like? What is it for? What are the benefits? And how to start?* (2014, Oct 5). Stack Overflow.https://stackoverflow.com/questions/5211746/what-is-cuda-like-what-is-it-for-what-are-the-benefits-and-how-to-start

12. Frąckiewicz, M. (2023, June 20). *The Benefits of Using Nvidia CUDA Toolkit for Computational Fluid Dynamics*. TS2 SPACE. https://ts2.space/en/the-benefits-of-using-nvidia-cuda-toolkit-for-computational-fluid-dynamics/

13. Schaeufele, J. (2009, February 22). *IT 103: Information Technology: NVIDIA and CUDA: Advantages and Disadvantages*. IT 103. http://jschaeuf.blogspot.com/2009/02/nvidia-and-cuda-advantages-and.html

14. *Thrust - Parallel Algorithms Library*. (2013, May 5). Thrust.github.io. https://thrust.github.io

15. *Visual Studio IDE, Code Editor, Azure DevOps, & App Center*. (2018, July 20). Visual Studio. https://visualstudio.microsoft.com

5.0 Appendix

Name(s): Hue Zhen Wei, Lee Wee Harn, Ricky Hwong Yu Jun Programme: RSW Group: G6 Date: 21/09/2023

Program (60%)

No	Item	Criteria			Final Marks
		Poor	Accomplished	Good	
1	**Output (10)**	• Inadequate information/ outputs needed are generated. • Most of the information/ outputs generated are less accurate. • Results visualisation is overly cluttered or the design seems inappropriate for the problem area. • Lack of information that are useful for the user 0 - 4	• Adequate information/ outputs needed are generated. • The information/ output generated are accurate but some with errors. • Pleasant looking, clean, well-organised results visualisation • The information displayed is useful for the user, but some details are omitted. 5 - 7	• All the necessary information/ outputs are generated. • All or most of the information/ outputs generated are accurate. Minor errors can be ignored. • The results are visually pleasing and appealing. • Great use of colours, fonts, graphics and layout. • The information displayed is useful to the users and complete with necessary details. 8 - 10	
2	**Programming (10)**	• The end product fails with many logic errors, many actions lacked exception handling. • Solutions are over-simplified. • Programming skills need improvement. 0 - 4	• Major parts are logical, but some steps to complete a specific job may be tedious or unnecessarily complicated. • Program algorithm demonstrates an acceptable level of complexity. • The student is qualified to be a programmer 5 - 7	• Correct and logical flow, exceptions are handled well. • Demonstrates appropriate or high level of complex algorithms and programming skills. 8 - 10	
3	**Degree of completion (10)**	• Too much still remains to be done. • Basic requirements are not fulfilled. • The end product produces enormous errors, faults or incorrect results. 0 - 4	• All required features present in the interface within the required scope, but some are simplified. • Or one or two features are missing. • The system is able to run with minor errors. 5 - 7	• All required features present in the interface • within or beyond the required scope. • No bugs apparent during demonstration. 8 - 10	
4	**Program Model Optimization (10)**	• The model is not optimized. • Most of the processes are executed in serial. • Only 1 parallel program model is used. 0 - 4	• The model is optimized by using more than 1 parallel program model, e.g., SPMD, loop parallelism. 5 - 7	• The model is optimized by using more than 1 parallel program model, e.g., SPMD, loop parallelism. • The model is tested on different parallel platforms, e.g., OpenMP, CUDA, MPI etc, in homogeneous or heterogeneous. 8 - 10	
5	**System implementation (10)**	• The end product is produced with a different system design or approach, which is not related to the initial proposal.	• The end product conforms to most of the system design, but some are different from the specification.	• The end product fully conforms to the proposed system design.	

		0 - 4	5 - 7	8 - 10	
6	**Presentation (10)**	• The student is unclear about the work produced, sometimes not even knowing where to find the source code.	• The student knows the code whereabouts, but sometimes may not be clear why the work was done in such a way.	• The student is clear about every piece of the work done.	
		0 - 4	5 - 7	8 - 10	
				Sum of Score	

Final Report (40%)

No	Item	Criteria				Final Marks
		Missing or Unacceptable	Poor	Accomplished	Good	
1	**Title and abstract (10)**	● Title or abstract were omitted or inappropriate given the problem, research questions and method.	● Title or abstract lacks relevance or fails to offer appropriate details about the education issue, variables, context, or methods of the proposed study.	● Title and abstract are relevant, offering details about the proposed research study.	● Title and abstract are informative, succinct, and offer sufficiently specific details about the educational issue, variables, context, and proposed methods of the study.	
		0 - 2	3 - 5	6 - 8	9 - 10	
2	**Results (Performance measurement) (10)**	● Analytical methods were missing or inappropriately aligned with data and research design. ● Results were confusing.	● Analytical methods were identified but the results were confusing, incomplete or lacked relevance to the research questions, data, or research design.	● The analytical methods were identified. ● Results were presented. ● All were related to the research question and design. ● Sufficient metric or measurement is applied.	● Analytical methods and results presentations were sufficient, specific, clear, structured and appropriate based on the research questions and research design. ● Extra metric or measurement is applied.	
		0 - 2	3 - 5	6 - 8	9 - 10	
3	**Discussion and Conclusion (10)**	● Discussions or answers to the research question and system performance were omitted or confusing. ● No or very little conclusion could be yielded.	● Little discussions were presented. ● Answers to the research question and system performance were unclear or confusing.	● Discussions of the results were presented. ● The research question and system performance were answered and identified.	● The significance of the results of the work was discussed, sufficiently inclusive of the information that concluded and answered the research question and system performance is evaluated comprehensively. ● Limitations and future improvements of the studies were identified.	
		0 - 2	3 - 5	6 - 8	9 - 10	
4	**Organization (5)**	● The structure of the paper was incomprehensible, irrelevant, or confusing. ● Transition was awkward.	● The structure of the paper was weak. ● Transition was weak and difficult to understand.	● A workable structure was presented for presenting ideas. ● Transition was smooth and clear.	● Structure was intuitive and sufficiently inclusive of important information of the research. ● Transition from one to another was smooth and organised.	
		0 - 2	3	4	5	
5	**Spelling, Grammar and Writing Mechanics (5)**	● There were so many errors that meaning was obscured, make the content became difficult to understand	● Some grammar or spelling errors were spotted. ● Some sentences were awkwardly constructed so that the reader was occasionally distracted.	● There were occasional errors, but they did not represent a major distraction or obscure meaning.	● Sentences were well-phrased. The writing was free or almost free of errors.	
		0 - 2	3	4	5	
					Sum of Score	

Final score = sum of scores/100*60 (base 60%)	