Parallel Implementation of the A* Pathfinding Algorithm on a Purely-Functional Programming Language

Prince Bernie B. Colis
Bachelor of Science in Computer Science
John Kenneth S. Lesaba
Bachelor of Science in Computer Science
Jon Ariel N. Maravilla
Bachelor of Science in Computer Science
Karl Frederick R. Roldan
Bachelor of Science in Computer Science

Senior project submitted to the faculty of the
Department of Computer Science
College of Computer Studies, Ateneo de Naga University
in partial fulfillment of the requirements for their respective
Bachelor of Science degrees

Month Day, 2020
Naga City, Philippines

Keywords: parallel programming, functional programming, graph theory

The Senior Project entitled

Parallel Implementation of the A* Pathfinding Algorithm on a Purely-Functional Programming Language

developed by

Prince Bernie B. Colis
Bachelor of Science in Computer Science

John Kenneth S. Lesaba
Bachelor of Science in Computer Science

Jon Ariel N. Maravilla
Bachelor of Science in Computer Science

Karl Frederick R. Roldan
Bachelor of Science in Computer Science

and submitted in partial fulfillment of the requirements of their respective Bachelor of Science degrees has been rigorously examined and recommended for approval and acceptance.

First Panel Member	**Second Panel Member**
Panel Member	Panel Member
Date signed: ______________________	Date signed: ______________________
Third Panel Member	**Adrian Leo Pajarillo**
Panel Member	Project Advisor
Date signed: ______________________	Date signed: ______________________

The Senior Project entitled

Parallel Implementation of the A* Pathfinding Algorithm on a Purely-Functional Programming Language

developed by

Prince Bernie B. Colis
Bachelor of Science in Computer Science

John Kenneth S. Lesaba
Bachelor of Science in Computer Science

Jon Ariel N. Maravilla
Bachelor of Science in Computer Science

Karl Frederick R. Roldan
Bachelor of Science in Computer Science

and submitted in partial fulfillment of the requirements of their respective Bachelor of Science degrees is hereby approved and accepted by the Department of Computer Science, College of Computer Studies, Ateneo de Naga University.

Marianne P. Ang, MS
Chair, Department of Computer Science
Date signed: ________________________

Joshua C. Martinez, MIT
Dean, College of Computer Studies
Date signed: ________________________

Declaration of Original Work

We declare that the Senior Project entitled

Parallel Implementation of the A* Pathfinding Algorithm on a Purely-Functional Programming Language

which we submitted to the faculty of the

Department of Computer Science, Ateneo de Naga University

is our own work. To the best of our knowledge, it does not contain materials published or written by another person, except where due citation and acknowledgement is made in our senior project documentation. The contributions of other people whom we worked with to complete this senior project are explicitly cited and acknowledged in our senior project documentation.

We also declare that the intellectual content of this senior project is the product of our own work. We conceptualized, designed, encoded, and debugged the source code of the core programs in our senior project. The source code of third party APIs and library functions used in my program are explicitly cited and acknowledged in our senior project documentation. Also duly acknowledged are the assistance of others in minor details of editing and reproduction of the documentation.

In our honor, we declare that we did not pass off as our own the work done by another person. We are the only persons who encoded the source code of our software. We understand that we may get a failing mark if the source code of our program is in fact the work of another person.

Prince Bernie B. Colis
3 - Bachelor of Science in Computer Science

John Kenneth S. Lesaba
3 - Bachelor of Science in Computer Science

Jon Ariel N. Maravilla
3 - Bachelor of Science in Computer Science

Karl Frederick R. Roldan
3 - Bachelor of Science in Computer Science

This declaration is witnessed by:

Adrian Leo Pajarillo
Project Advisor

Parallel Implementation of the A* Pathfinding Algorithm on a Purely-Functional Programming Language

by

Prince Bernie B. Colis, John Kenneth S. Lesaba, Jon Ariel N. Maravilla, and Karl Frederick R. Roldan

Project Advisor: Adrian Leo Pajarillo

Department of Computer Science

EXECUTIVE SUMMARY

To be filled in later. `/*TODO*/`.

I dedicate this research work to all of humanity.

ACKNOWLEDGEMENTS

I thank everyone who helped me finish this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

The A* pathfinding algorithm is a best-first pathfinding algotithm for graphs commonly used for graph traversal applications such as artificial intelligence, video games, flight paths, and more. However, while most games are written in an imperative and object-oriented language such as C#, C++, and JavaScript, it is possible to write video games in a functional language using a reactive functional programming approach.[6] Likewise, the need for other correct critical software led organizations such as NASA to use Haskell[1], a purely-functional programming language, to be used in systems where high-level assurance and provable programs are a must.[25]

This paper assumes concrete differences between *parallel* and *concurrent* where the former is defined to be a hardware feature of having multiple processors or cores to compute a problem whereas the latter is defined to be a software-based approach to decrease the impact of computation bottlenecks by switching between different computations when a computation takes too long.[30] One of the major challenges of parallel programming is controlling the order of execution to prevent *race conditions*, which can often lead to bugs and are hard to maintain. However, since pure functional languages, such as Haskell, have no mutability and computations lead to the same result regardless of the order, they are a perfect candidate for writing parallel programs.[13] This research aims to find a parallel implementation of the existing A* pathfinding algorithms using a purely-functional setting with attention to program performance. In turn, this helps in the advancement of different functional programming approaches for parallel graph computations which could eventually lead to critical systems to use a more provable programming language.

1.1 Project Context

The current video game industry is dominated by imperative programmers, even though functional programming had been getting more popular recently especially in web-programming due to to languages such as TypeScript and PureScript and frameworks such as ReactJS which favors a reactive functional approach. Thus, most beginners are exposed to imperative algorithms, such as the A*, due to the popularty of algorithm books with imperative programming in them.[7, 20, 31] However, algorithms designed for imperative programming may not translate as well when implemented in a functional style. The quicksort specification maintains a worst-case space complexity of $O(n)$. However, when the algorithm is written in a pure-function style, it will have to copy every subarray due to its recursive nature, thus yielding a worst-case space complexity of $O(n^2)$.

The advantages of functional programming lies with its *referential transparency* which means that a function definition or a variable will never change its definition throughout the runtime of the program.[18, 13] Hence, mathematically proving functional programs might be easier and can be aided by proof assistants such as Coq or Agda.[4, 32, 9] Likewise, splitting functions into smaller functions and reasoning about those smaller components much like lemmas would mean that functions would be modular and composed of proven subfunctions.[3, 16] Hence, functional programming languages are excellent candidates for parallel programming since the languages do not have mutable states and are therefore, instances of shared variables are abstracted away from the programmer. Similarly, the order of execution of pure functions does not matter as the program will still yield the same results.[18] A parallel and purely-functional approach to the A* algorithm would eventually lead to more applications such as shortest distance in a map, flight paths, web server searching, and more to use a more provable and type-safe language which could lead to less system failures and high availability.

1.2 Purpose and Description

This research aims to utilize the existing parallel A* pathfinding algorithm [11, 33] and find a way to develop a reasonably-efficient purely-functional implementation of the algorithm using parallel data structures such as STMs or MVars[23].

The A* Pathfinding algorithm is used heavily in video games, telephone traffic, and other graph traversal problems[15]. This research aims to aid in the development of video games and other uses where a parallel A* algorithm would be beneficial using the functional programming paradigm in the future as video game development is dominated by imperative languages.

1.3 Objectives

The main objective of the research is to find an efficient parallel purely-functional implementation of the A* pathfinding algorithm. The research will be done mostly in Haskell with some exceptions. Likewise, concrete comparisons between the number of cores and logical threads will be used to measure the most efficient performance runtime and space complexity of the algorithm.

The researchers aim to complete the following specific tasks:

- Write a *generator* that will generate an arbitrary-sized maze. The maze should be relatively hard to solve without the aid of computers in a short amount of time.[5]
- And a *solver* program that will be written in Haskell, a lazy purely-functional programming language, for translating the output of the generator to a graph.[1]
- The solver program should have a web-based user interface where the researchers can view the maze and how the solver program was able to solve it correctly.
- Performance of the solution shall be measured by using ThreadScope to monitor the thread and core activities while the program is being run.[2]

1.4 Scope and Limitations

The research will only cover solvable-mazes as the A* algorithm does not halt when there is no reachable end goal (e.g, the start vertex and end vertex lie on different components of the graph).[15] Likewise, there will be no generality and all programs will be written in Haskell. Translation to other functional programming languages is not a priority and, thus, lambda notation will not be used. Other concurrent data structures besides MVar and Software Trasnactional Memory will not be utilized. The implementation of the graph that the research will use will be Algebraic Graphs.[24]

The concrete implementation and analysis is planned to be tested only on four CPUs such as Intel Core i7-9750H and AMD Ryzen 5 3500x. Other CPU architectures are not planned to be tested on.

Chapter 2

Review of Related Systems and Related Literature

Due to the trend of parallelization in the modern computing era, [28, 33, 11, 29] several researchers developed more efficient and faster implementations of the A* algorithm by parallizing the algorithm in various ways.

2.1 The A* Algorithm

Here, we shall describe the original A* algorithm by Hart, Nilsson, and Raphael.[15] A pure-function pseudocode1 will be implemented instead of the usual imperative pseudocode. But the pseudocode still follows the specifications of the original paper.

Algorithm 1 Sequential A* Algorithm

```
Input: G - a weighted labeled graph.
Input: (ψ, ξ) - a tuple of two sets: the open set and the closed set
Input: h - the heuristic function. h(n) ≠ 0 or this equates to Dijkstra's Algorithm.
Input: (σ, γ) - a tuple of two vertices in G: the start vertex and end vertex.
Output: p - The shortest path from ψ to ξ.
  let S be a homogenous set of vertices in a graph.
  astar :: Graph → (S, S) → ((Int, Int) → (Int, Int) → Int) → (Vertex, Vertex) → Path
  astar g s h se p = WRITE THE ALGORITHM LATER.
```

2.2 Parallel A* Algorithm

Here, we shall discuss the naive implementation of Zaghloul[11] where the researchers parallelized A* using MISD and compared the running times and speed up of the algorithm in solving some problems based ont he number of cores of the computer it was run on.

Also, we shall visit the implementations by Rios and Chaomowicz[28] which used a more enhanced implementation of the single-core bidirectional pathfinding algorithm by Kaindl and Kainz[17] by parallelizing the two directions on two CPU cores.

Likewise, we shall also visit HDA* by Kishimoto, Fukunaga, and Botea[19] on how it assigned ownership of vertices between different CPU cores by hashing and the performance comparison of Holladay and Weinstock[33].

Lastly, we compare how the HDA* may be a problem for Haskell since Haskell's parallelism is abstracted. Instead, the paper will define another method of parallelizing the A* algorithm on a multicore machine using Haskell.

2.3 Inductive Graphs

This section will discuss the `functional graph library` as described by Erwig.[10] Inductive graphs can be constructed monadically which will be of advantage when writing parallelizing an algorithm since parallel code can only be executed monadically.

We use both inductive graphs for the sequential implementation (not a monad) and for the parallel implementation (monad).

2.4 Real-time Performance of Functional Programs

In this section, we compare the usefulness of functional programs in real-time situations, that is, in applications were time is critical such as GPS systems and flight guidance software.[12] Since we're primarily concerned with developing a fast-enough parallel program for a correct software that would be needed for critical systems, we have to compare the performance of our program with recent imperative implementations. This should be stated here.

Likewise, we make use of Tim Harris, Simon Marlow, and Simon Peyton Jones' research on the parallel performance of Haskell and overhead costs when using parallel data structures.[14]

Appendix A

Code Listing

REFERENCES

[1] *Haskell programming language*. `https://haskell.org`.

[2] *Threadscope*. `https://github.com/haskell/ThreadScope`.

[3] A. Abel, M. Benke, A. Bove, J. Hughes, and U. Norell, *Verifying haskell programs using constructive type theory*, 01 2005, pp. 62–73.

[4] J. Breitner, A. Spector-Zabusky, Y. Li, C. Rizkallah, J. Wiegley, and S. Weirich, *Ready, set, verify! applying hs-to-coq to real-world haskell code*, 2018.

[5] J. Buck, *Mazes for Programmers*, The Pragmatic Programmers, 2015.

[6] M. H. Cheong, *Functional programming and 3d games*, (2006).

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, The MIT Press, 3rd ed., 2009.

[8] L. Dhulipala, G. E. Blelloch, and J. Shun, *Theoretically efficient parallel graph algorithms can be fast and scalable*, in Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA '18, New York, NY, USA, 2018, Association for Computing Machinery, p. 393–404.

[9] Y. El Bakouny, T. Crolard, and D. Mezher, *A coq-based synthesis of scala programs which are correct-by-construction*, Proceedings of the 19th Workshop on Formal Techniques for Java-like Programs, (2017).

[10] M. Erwig, *Inductive graphs and functional graph algorithms*, Journal of Functional Programming, 11 (2001), pp. 467–492.

[11] Z. et al., *Parallelizing a* path finding algorithm*, International Journal Of Engineering And Computer Science, 6 (2017), pp. 22469–22476.

[12] S. Frame and J. W. Coffey, *A comparison of functional and imperative programming techniques for mathematical software development*, Journal of Systemics, Cybernetics and Informatics, 12 (2014), pp. 1–10.

[13] K. Hammond, *Why parallel functional programming matters: Panel statement*, in Reliable Software Technologies - Ada-Europe 2011, A. Romanovsky and T. Vardanega, eds., Berlin, Heidelberg, 2011, Springer Berlin Heidelberg, pp. 201–205.

[14] T. Harris, S. Marlow, and S. P. Jones, *Haskell on a shared-memory multiprocessor*, in Proceedings of the 2005 ACM SIGPLAN Workshop on Haskell, 2005, pp. 49–61.

[15] P. E. Hart, N. J. Nillson, and R. Betram, *A formal basis for the heuristic determination of minimum cost paths*, IEEE Transactions of Systems Science and Cybernetics, SSC-4 (1968).

[16] J. Hughes, *Why Functional Programming Matters*, Computer Journal, 32 (1989), pp. 98–107.

[17] H. Kaindl and G. Kainz, *Bidirectional heuristic search reconsidered*, Journal of Artificial Intelligence Research, 7 (1997).

[18] M. Kesseler, *The Implementation of Functional Languages on Parallel Machines with Distributed Memory*, PhD thesis, Radboud University Nijmegen, 1996.

[19] A. Kishimoto, A. Fukunaga, and A. Botea, *Scalable, parallel best-first search for optimal sequential planning*, Association for the Advancement of Artificial Intelligence, (2009).

[20] D. E. Knuth, *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*, Addison Wesley Longman Publishing Co., Inc., USA, 1997.

[21] H.-W. Loidl, F. Rubio, N. Scaife, K. Hammond, S. Horiguchi, U. Klusik, R. Loogen, G. J. Michaelson, R. Peña, S. Priebe, Á. J. Rebón, and P. W. Trinder, *Comparing parallel functional languages: Programming and performance*, Higher-Order and Symbolic Computation, 16 (2003), pp. 203–251.

[22] A. Lumsdaine, D. P. Gregor, B. Hendrickson, and J. W. Berry, *Challenges in parallel graph processing*, Parallel Process. Lett., 17 (2007), pp. 5–20.

[23] S. Marlow, *Parallel and Concurrent Programming in Haskell*, O'Reilly Media, Inc., 2013.

[24] A. Mokhov, *Algebraic graphs with class (functional pearl)*, SIGPLAN Not., 52 (2017), p. 2–13.

[25] I. Perez, F. Dedden, and A. Goodloe, *Copilot 3*, Tech. Rep. 20200003164, National Aeronautics and Space Administration, May 2020.

[26] R. C. Prim, *Shortest Connection Networks And Some Generalizations*, Bell System Technical Journal, 36 (1957), pp. 1389–1401.

[27] F. Rabhi and G. Lapalme, *Algorithms; A Functional Programming Approach*, Addison-Wesley Longman Publishing Co., Inc., USA, 1st ed., 1999.

[28] L. Rios and L. Chaimowicz, *Pnba * : A parallel bidirectional heuristic search algorithm*, 2011.

[29] V. Sanz, A. D. Giusti, and M. Naiouf, *Improving hash distributed a* for shared memory architectures using abstraction*, in ICA3PP, 2016.

[30] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, Wiley Publishing, 10th ed., 2018.

[31] S. S. Skiena, *The Algorithm Design Manual*, Springer, London, 2008.

[32] A. Spector-Zabusky, J. Breitner, C. Rizkallah, and S. Weirich, *Total haskell is reasonable coq*, Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, (2018).

[33] A. Weinstock and R. Holladay, *Hash distributed a* distributed termination detection path reconstruction*, 2016.

VITA

`/*TODO*/` are BS Computer Science student of the Department of Computer Science at the Ateneo de Naga University.