Parallelized Traffic Simulation in C++

Zach Alaniz
Department of Computer Science
College of Science and Engineering
Texas Christian University
Fort Worth, Texas 76129
Email: z.a.alaniz@tcu.edu

Saby Sahoo
Department of Computer Science
College of Science and Engineering
Texas Christian University
Fort Worth, Texas 76129
Email: s.sahoo@tcu.edu

Bradley Schoeneweis
Department of Computer Science
College of Science and Engineering
Texas Christian University
Fort Worth, Texas 76129
Email: b.schoeneweis@tcu.edu

***Abstract*—This semester research project investigates the potential speedups and scalability of running a traffic simulation in parallel using C++ and OpenMP. These speedups are evaluated by comparing the sequential performance to the parallel performance when varying thread counts for a fixed-size, randomized input. While a large percentage of the program is inherently sequential due to computation, the process of transitioning from one state to another was parallelized.**

***Index Terms*—OpenMP, Cellular Automata, Parallel Simulation, C++**

I. Introduction

Extensive research and improvements have been made over the years in the field of study that is parallel computing. Falling within this domain of study is the topic of distributed simulation. As models become more and more complex, the execution time associated with these simulations becomes far too substantial for a sequential approach. A common remedy to this issue is to redesign the simulation to run in parallel. This research aims to see the benefits of running a parallelized discrete event simulation, and more specifically, a parallelized traffic simulation, over a sequential simulation. [1]

The ability to optimize light scheduling and move cars from one point to another in an efficient, organized manner is paramount for day-to-day travel. Being able to apply parallel computing to simulate increasingly large traffic environments leads to many interesting research topics and advances in road safety, traffic capacity, economic impact, among many others. The topic of simulating traffic flow for increasingly large sizes of roads is the topic of interest explored in this simulation. Real life traffic flow allows for movement when possible at all times, and by utilizing parallel computing paradigms and tools, the speedup of a traffic simulation should prove beneficial in seeing how a particular road and intersection scales for an increasingly large number of cars funneling in and out.

Traffic simulation naturally lends itself to parallel computing paradigms. Decomposing the movement of cars into a sub-domain of problems and updating traffic flow in parallel from one state to another, just as normal traffic would run, is one method to improved performance. Thus, a parallel solution to traffic flow needs to be able to parallelize any movement and update that could be happening concurrently. The dynamic nature of traffic also plays a large role in this type of simulation but was handled to leave a larger emphasis on the updating of car movements. The following research being presented simulates traffic flow through a four-way stop, which includes two-lane roads from all directions. The traffic simulation was built with C++ utilizing a cellular automaton approach. The parallelization was made with OpenMP, an application programming interface (API) that supports shared memory multiprocessing. The following is an overview of the remainder of the paper: Background research on the applied tools and ideas used and reasons for why they were chosen, the sequential algorithm design and solution, the parallelization and results, and finally a summary with conclusions of the research.

II. Background and Related Research

Chances are if you have ridden in a vehicle, you have come across a four-way stop being controlled by traffic lights. A four-way traffic stop has roads leading from all four directions (North, South, East, West) where each direction takes a turn moving towards its destination as directed by the controlling traffic light. The traffic light operates in three states: red, green, and yellow. Green means go, red means stop, and yellow means slow down. Assuming each direction is at least a two-lane

road, and ignoring the possibility for a one-way street, any car driving on the outer most lane is capable of taking a right turn at the traffic light intersection if oncoming traffic is clear, regardless of the state of the light. Cars in the innermost lanes are also capable of making a left turn, given that their light is currently green. The light itself cycles on a timer allowing for each lane to have its own turn to proceed, which creates the flow of traffic. This four-way model is what our simulation is derived from, which will be explained in a later section.

The placement of cars along any road is relatable to a data model studied in computer science and related fields known as *cellular automata*. Formally, cellular automata are defined as a collection of cells on a grid that evolves during a number of discrete time steps according to a set of rules based on the states of neighboring cells. The rules are then applied iteratively for as many time steps as desired [2]. Cellular automata have been studied since the early 1950's and was initially used as a model for biological systems. The lattice structure and discrete state like approach of cellular automata was taken and applied to our simulation to represent individual, discrete states of traffic flow and positions of cars within a street. Another principle of cellular automatons is localized interactions with nearby cells. Each car within this system must recognize the state of the cell that lies a step ahead in the direction it wishes to move in order to determine its next state. The different discrete representations of the states and interactions of the cars, the lights, traffic, and lanes in general will be explained later within the paper.

The general structure of the system has been described, now let's look at the tools and technologies used to implement the simulation. The entirety of this research project was developed in C++ 17. C++ is a statically-typed, compiled, middle-level, general purpose programming language developed by Bjarne Stroustrup in 1979. C++ was used to implement all of the simulation logic and sequential transitions. To parallelize this program, we used OpenMP. OpenMP is an Application Program Interface (API) that provides a portable and scalable shared memory model for parallel applications. OpenMP is designed for shared memory parallel programming. This was essential for our project. What drove us to choose OpenMP over MPI (Message Passing Interface), which doesn't have a concept of shared memory, was the potential costs of communication if the simulation were to be run with an input size of 200,000,000. This would require an MPI implementation to have the processes broadcast the global automata system for every discrete state, which could be incredibly costly at scale. With OpenMP, this global system can be shared memory and operated on by all the threads, without the overhead of frequent, massive scale communication instructions for all processes to get back on the same page. Another common parallelization library is POSIX threads (or pthreads). We chose OpenMP over pthreads due to critical section concerns and keeping the threads in sync while accessing the shared data simultaneously. Pthreads offered too fine-grained control for the needed operations our system performs, and the flexibility given was overkill for this simulation's purpose. OpenMP is more high level than pthreads, and allowed for easier, more accurate division of tasks among processes. As a result, OpenMP was the parallelization tool of choice for this project . [3]

Now that the technologies and techniques have been selected to maximize our simulation specifications, a resource is needed for the execution, testing, and analysis of our program. In order to run high intensity simulations, for both thread count and input size, we would submit our project for execution as a batch job onto the Stampede2 computing cluster. Stampede2 is the flagship supercomputer at the Texas Advanced Computing Center, and consists of 1,736 nodes, and each has 48 cores and 192GB of RAM. This enabled us to run simulations with threads varying from one up to 64 threads on input sizes ranging from 200 to 200,000,000. [4]

With all resources in hand, we can discuss the history of traffic models. Parallelized traffic simulations have been implemented before and have evolved from relatively simple systems to very complex models. Here are some of the implementations that inspired the following research, and set the path for modern, parallel traffic simulations. One of the very first cellular automaton-based simulations was designed by Nagel and Schreckenberg. This simulation was developed for single lane traffic and was initially sequential. It utilized Monte-Carlo methods to describe start-stop waves as traffic grows more and more dense. While relatively simple, this model was used for traffic forecasts and analysis for years. As systems began to adapt multiple lanes of traffic and simulations were adapted to model high-density, urban areas, the need for parallelization quickly became realized. One of

the first thoughts for parallelization was decomposing the automaton into sub-systems. The issue with this approach was something we also discovered from testing, was that the communication overhead when the system scales becomes far too costly for efficient execution. Another approach that researchers sought out was dividing the system into preexisting time segments and eventually synchronizing the time slices. This also was not effective, as the outcomes of these time slices cannot be predetermined, as there must be an essence of randomicity to keep the simulation true. Modern traffic parallelization utilizes methods for accurate systematic predictions to decompose the system into micro-states and optimize the computation to set-up the next state of the system while load-balancing the system splits to prevent massive communication costs. [5]

Where this research project attempts to differentiate itself is by simplifying some of the logic of the random states and primarily focus on the parallelization of the state transition without decomposition. The hope is that at a large enough scale, our state transitions of even massive systems will be able to see performance improvements through OpenMP parallelization. To do this, we have essentially simplified the next state logic for our four-lane simulation to adhere to a transition function of simply shifting each of the lanes towards the next determined state (computed prior to shift time). The next section of this paper will discuss the sequential implementation of the traffic simulation, how the states are determined, and the operation to transition from one discrete state to another.

III. Parallelization Technique

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

IV. Summary of Results

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

V. Conclusions, Lessons Learned, and Future Work

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

References

[1] R. M. Fujimoto, "Parallel discrete event simulation," *Commun. ACM*, vol. 33, pp. 30–53, Oct. 1990.

[2] "Cellular automaton."

[3] B. Barney, "Openmp."

[4] "Texas advanced computing center."

[5] T. Kiesling and J. Luthi, "Towards time-parallel road traffic simulation," in *Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*, pp. 7–15, June 2005.