De La Salle University
College of Computer Studies
Software Technology Department

CSC615M

Technical Paper

Section	G02
Team Members	Fernandez, Ryan Austin
	Poblete, Clarisse Felicia M.
	Tan, Johansson E.
Date Submitted	December 9, 2016

Table of Contents

1. Introduction

In the world of computation, there is a need for a model for parallel processes. Since computation seeks to model the real world, it is natural to expect that computations in parallel be needed. This may include modelling complex concurrent interacting systems, wherein the state of a component of the system depends on the state of the neighboring components. This such computational model is commonly known as a cellular automata (Pokkuluri, K.S. & Inampudi, R. D.,2013). Other models include a model for regular expression matching where the input string can be split arbitrarily and the computation on each resulting substring can be done in parallel. This is known as simultaneous finite automata (Sin'ya, R., Matsuzaki, K., & Sassa, M.,2014).

This paper recognizes the vast topic of parallelism and the multiple approaches and issues in dealing with parallel computing models. The aim of this paper is to survey the current landscape of parallel computational models. Specifically, this paper aims to discuss the needs for parallel computational models, the approaches to parallel computing, including some existing models, and the issues involved in parallel computing.

The need for parallel computing will only cover models solvable by parallel modifications of various computational models discussed in the course. The approaches will only cover parallel computing pushdown automata, cellular automata, simultaneous finite automata, and asynchronous systems of parallel computing finite automata.

This topic's significance lies in the inherent nature of the real world to have multiple processes running simultaneously. Modelling such processes requires parallel computing, which can be handled by the parallel computing models discussed in this paper. The computational models discussed in class may not be sufficient to model these systems; a parallel model is required for such systems.

2. Review of Related Literature

Balan, M. (2009) discussed that a variant of parallel communicating pushdown automata (PCPA) called returning centralized PCPA has equivalent computing capabilities of the multi-head pushdown automata. PCPA consists of pushdown automata (PDA) working in parallel to accept a language. These automata communicate which each other via stacks wherein a certain query symbol is needed from the requesting component in order to transfer stack information of the requested component to the requesting component's stack. A property called known communication property was enforced on the PCPA variant which makes whatever communicating component know when the communication occurs at its state level. With this property defined in the returning centralized PCPA, it has been shown the equivalent communicating capabilities of it with multi-head pushdown automata.

Dourvas, N. I., Sirakoulis, G. C., & Tsalides, P. (2015) used a parallel computational model called Cellular Automata (CA) to model the computing abilities of slime mould called Physarum Polycephalum. Its plasmodium has been used for solving several problems on shortest path, graphs, and many engineering applications. A GPU implementation with CUDA programming was used for more parallel computation. The CA, in this case, is modelled to simulate to find the minimum path of the movement of the plasmodium on one spot in a maze going to another spot with food.

Kalogeropoulos, G., Sirakoulis, G., & Karafyllidis, I. (2013) used cellular automata (CA) for a model on traffic signals control and traffic flow. The CA model presented is characterized with low complexity in order to keep computational recourses low while keeping computation speed high without any loss of the complexity. The CA model aims to enrich available information on the current traffic flow and condition, and to provide traffic prediction and management.

Otto, F. (2015) introduced a modified version of parallel communicating systems of pushdown automata (PC systems of pushdown automata) that is asynchronous. PC systems of pushdown automata only make use of a query symbol in communication, but the modified version proposed also makes use of a response symbol. The results revealed that asynchronous PC systems of pushdown automata of degree n were capable of defining the same language classes as an n-head pushdown automata.

Pokkuluri, K.S. & Inampudi, R. D. (2013) modelled the prediction of protein structures from existing proteins. In order to do this, they used cellular automata combined with random generation of populations from an initial seed population involving single point crossover to progress the simulation. They achieved a 80% - 89.8% accuracy in predicting the protein structure depending on the dataset.

Sin'ya, R., Matsuzaki, K., & Sassa, M. (2014) proposed a machine called a simultaneous finite automaton (SFA). Given an arbitrarily long string, the string can be split up at any point and fed to the SFA in parallel. The result is the product of all the individual threads. The construction of an SFA from any finite automaton with n states is straightforward but may lead to the resulting machine having at most n^n states. In practice, however, SFA's only result in machines with around n^2 states compared to the original machine.

Vollweiler, M. (2015) introduced asynchronous systems of parallel communicating finite automata (APCFA), looked at the differences between synchronous and asynchronous systems and between blocking and non-blocking communication, and shared results of the communication complexity of APCFA. The results showed that only regular languages are accepted with constantly many communications for APCFA, while, with respect to input length, at most linearly many communications and polynomially many communications are needed for one-way and two-way components, respectively.

3. Discussion

3.1. The Need for Parallelism

Computation seeks to model some real-world process. The real world is a complex system with multiple interleaving components. The state of one component can affect the change in the state of another component. Parallelism can help model these kinds of systems or simply increase the efficiency of a computational system.

An example of a complex system that can be modelled by a parallel computational model is protein structures. The structure of an amino acid is defined by its own state, called the first generation technique, but also with the state of its neighboring environment, called the second generation technique (Pokkuluri, K.S. et al.,2013). In such a case, the system is too complex to be effectively modelled as a sequential system. It would be more appropriate to model such an environment with a parallel computational model such as a cellular automata. Additional information about cellular automata will be further discussed in the next section.

In regular expression matching for arbitrarily long strings, it may be desirable to decrease the running time. This can be theoretically done by splitting the string into multiple substrings then running the substrings on the machine in parallel, getting the boolean product of the resultant machine runs. If all runs are accepted, the string is accepted. This was modelled as the simultaneous finite automata (Sin'ya, R., et al.,2014). Given the ideal number of threads based on the machine, this would make regexp pattern matching run faster.

Parallel computational models are needed to model complex parallel systems or in order to decrease the runtime of otherwise sequential computational models. There are multiple existing computational models that will be further discussed in the next section.

3.2. Parallel Computational Models

3.2.1. PCFA and PCPDA

A parallel communicating (PC) system is composed of several components that work on their own but run together in parallel. For PC finite automata (PCFA), each of these components is a finite automata, while for PC systems of pushdown automata (PCPDA), each of these components is a pushdown automaton. Synchronized PC systems run stepwise, meaning that they each perform one communication step in one unit of time. Since they run at the same speed, they are considered to be synchronized. In contrast to this, asynchronous systems may involve an arbitrarily long period of waiting, meaning that the components may not proceed completely stepwise in parallel with one another.

For example, for synchronous communication, when a component *x* communicates with another component *y*, *x* initiates the communication. For PCFA, this occurs when *x* enters a communication state, while for PCPDA, this occurs when a query symbol is encountered at the top of the stack. After this, the communication step occurs immediately. In synchronous systems, this communication step is when, for PCFA, *y* sends its current state to *x* and the finite control of x is set into that state, or, for PCPDA, the query symbol is replaced by the complete pushdown content of *y*. When this occurs, *y* is not actually aware of *x* nor that the communication even took place. This is why this communication is also considered to be one-directional.

With asynchronous communication, on the other hand, the communication is bidirectional, i.e. both components involved are aware of the communication components and time, and both components need to be ready for the communication before it can take place.

To allow for this, the asynchronous parallel communication for PCFA presented by Vollweiler makes use of four different communication states, rather than just one, namely, the request, response, receive and acknowledge states. Using this protocol, if a component *x* communicates with another component *y*, it enters a request state to request information from *y*. Conversely, if *y* is ready to communicate with *x*, it enters a response state to transmit information from *y* to *x*. If *x* is in a request state, but *y* is not yet in a response state, *x* waits, and vice versa. All other components continue independently while these components wait. Once *x* is in a request state and *y* is in a response state, the communication step occurs, wherein *x* moves to a receive state, receiving the information transmitted from *y*, and *y* moves to an acknowledgment state, acknowledging the receipt of the message. After this, the components continue independently.

In the same way that asynchronous PCFA make use of extra states, asynchronous PCPDA make use of an additional symbol called the response symbol. As with the synchronous PCFA, a component *x* communicates with another component *y* by placing a query symbol at the top of the stack. However, if there is no response symbol at the top of the stack of *y*, then *x* waits until the response symbol is there before proceeding to the communication step. The same goes for *y*, which places a response symbol at the top of its stack when it’s ready to communicate, but waits for a query symbol to appear at the top of *x* before moving on to the communication step.

Asynchronous systems may make use of either blocking or non-blocking communication. The protocol presented made use of blocking, wherein components in the process of communicating are blocked from continuing with their own independent computations until they are finished with their current communication. Non-blocking, on the other hand, does not wait until communication and the processing between the involved components is done before proceeding independently. Even in deterministic machines, the same input may lead to different computations for non-blocking communications. (Otto, 2015; Vollweiler, 2015)

3.2.2. Cellular Automata

Cellular automata (CA) are used to model physical systems as the collective effects of locally interacting components of the system gives rise to the global behavior of certain features in the system. CA used as they are best suited for hardware realization especially the straightforward implementation for the automata.

There have been works that have utilized CA to simulate certain real-world processes. In Dourvas, N. I., Sirakoulis, G. C., & Tsalides, P. (2015), the computing ability of a slime mould called Physarum Polycephalum was modelled using CA. The CA model is computed as follows. First, the next state of each of the cells in the CA is computed in parallel. The storing of this information is also done in parallel. Lastly, the switch between the current state to the next state of each cell also takes place in parallel. For the simulation of the slime mould in the maze, the maze was represented with a 50x50 CA cells. In every time step (the whole process for the next state), the plasmodium as well as chemo-attractants from the food source spreads through the maze in directions it can possibly go to. There will come the time when the plasmodium that spread meets with the chemo-attractant. At this point, the plasmodium will spread to the path of the chemo-attractant that has the greatest quantity of it. The greater the quantity of the chemo-attractant means less the distance it took to spread. The plasmodium will spread til it reaches the food source and at the same time reduces its spread of the plasmodium on other paths.

The parallelism displayed by cellular automata in this work shows the computing capabilities of these kind of automata to solve different kinds of problems such as the one showcased here which is the shortest path problem.

Another work that used CA to model a real life process is in the work of Kalogeropoulos, G., Sirakoulis, G., & Karafyllidis, I. (2013). This work delves on using CA to model traffic control and flow in the real world. The cellular automaton was designed to have cells being all 4-way road intersections with equal lengths with one another. The time step to change states involves information about the flow that will come from every direction which will indicate as to what state will be next for each cell. The end result is CA model focused on strategies for traffic flow control and finding proper parameters to maximize the flow.

These works have shown how parallelism in automata models, in this case the one in cellular automata, is beneficial in modelling complex real life processes which regularly involves simultaneous actions or tasks. Using these kinds of automata in real life process does present some issues. Certain limitations are brought up when the automata is implemented.

3.3. Issues and Limitations

One such limitation is the hardware specifications that would be needed in implementing such automata. In Dourvas, N. I., et al. (2015), in order to exploit the parallelism of cellular automata, a Graphical Processing Unit (GPU) implementation was done. The GPU here is implemented with the CUDA programming model. CUDA provides 3 key abstractions, namely: organization of threads, organization of memory and functions that execute in parallel. As such, the parallel nature of cellular automata is made possible because of CUDA. However, this is only made to work with certain hardware available, in this case, the graphics card NVIDIA GT640 was used. The same hardware constraint existed in the work of Kalogeropoulos, G., et al. (2013) but in this case a field-programmable gate array (FPGA) was used.

In PCPDA, a certain issue potentially arises which is the communication of the components within this kind of automata. At the state level of the communicating component, the communicating component does not have information on when the communication occurs. This notion for component communication appears in Balan, M. (2009). In the process of showing that returning centralized PCPDA had the same computing capabilities as a multi-head pushdown automata, such constraints were encountered. As such, a property was forced into the PCPDA which was called the known communication property. This property makes known to the communicating component when the communication occurs at the state level.

These are only some of the issues that are possibly encountered in exploring parallel automata. Other past works may contain more issues with regards to parallel automata. This does not necessarily diminish the power and capability that comes in utilizing the parallelism in these computational models.

4. Conclusion

The complexity of the real world provides the need for parallel computational models. It is simply insufficient to say that a sequential approach is applicable for all models.

There are already various computational models meant to model parallel systems such as cellular automata, parallel communicating automata, both finite and pushdown, and simultaneous finite automata. Multiple studies have been mentioned that have made good use of these computational models.

However, some issues may arise such as hardware limitations decreasing the degree of parallelism in terms of physical processing, or poor coordination between modules in the parallel communicating automata.

Parallelism is still a viable option in modelling the real world. As has been seen in this paper, existing computational models have been efficiently used to model the real world and it is not far-fetched to think that greater advancements along this line could come about in the following years.

References

Balan, M. S. (2009). *Serializing the parallelism in parallel communicating pushdown automata systems*. doi:10.4204/EPTCS.3.5

Denning, P.J., Dennis, J.B., & Qualitz, J.E. (1978). *Machines, languages, and computation*. New Jersey: Prentice Hall.

Dourvas, N. I., Sirakoulis, G. C., & Tsalides, P. (2015). *GPU implementation of Physarum cellular automata model*. AIP Conference Proceedings, 1648(1), 1-4. doi:10.1063/1.4912827

Hopcroft, J.E., Motwani, R., and Ullman, J. D., (2006), *Introduction to automata theory, languages and computation*. 3rd edition, Addison-Wesley Publishing Company. Reprinted by Jemma Inc.

Kalogeropoulos, G., Sirakoulis, G., & Karafyllidis, I. (2013). *Cellular automata on FPGA for real-time urban traffic signals control*. Journal Of Supercomputing, 65(2), 664-681. doi:10.1007/s11227-013-0952-5

Otto, F. (2015). *Asynchronous parallel communicating systems of pushdown automata*. International Journal Of Foundations Of Computer Science, 26(5), 643-666.

Pokkuluri, K. S., Inampudi, R. B., & Usha, D. N. (2013). *An extensive report on cellular automata based artificial immune system for strengthening automated protein prediction*.

Romani, F. (1978). *The parallelism principle: speeding up the cellular automata synchronization*. Information and Control, 36(3), 245-255, ISSN 0019-9958

Sin'ya, R., Matsuzaki, K., & Sassa, M. (2014). *Simultaneous finite automata: An efficient data-parallel model for regular expression matching*.

Vollweiler, M. (2015). Asynchronous Systems of Parallel Communicating Finite Automata. *Fundamenta Informaticae*, *136*(1/2), 177-197. doi:10.3233/FI-2015-1149.